



DIMS User Manual Documentation

Release 0.2.1

Dave Dittrich

Mar 26, 2017

Contents

1	Introduction	3
1.1	Introduction	3
2	Referenced documents	5
3	Development and Core Tool Policy	7
3.1	General Software Development Philosophy	7
3.2	Source Code Control	9
3.3	Copyright	9
3.4	License	9
3.5	Developing on a fork from GitHub	10
3.6	Developing	11
4	Source Code Management with Git	13
4.1	Foundational Git Resources	14
4.2	The need for policy and discipline	14
4.3	Global Git Configuration	17
4.4	Daily tasks with Git	17
4.5	Infrequent tasks with Git	25
5	Documenting DIMS Components	57
5.1	Required Background Reading	57
5.2	Why Sphinx?	57
5.3	Manually Initiating a docs directory with sphinx-quickstart	58
5.4	Building Sphinx Documentation	61
5.5	Fixing errors	67
5.6	Common Tasks	75
5.7	Common Problems	78
5.8	Advanced Use of Sphinx Features	82
6	Continuous Integration	91
6.1	Continuous Integration	91
6.2	How source changes are propagated	91
6.3	Continuous deployment of documentation	92
7	Developing modules for the DIMS CLI app (dimsccli)	99
7.1	Bootstrapping the dimsccli app for development	99

7.2	Command Structure	103
7.3	Completing commands in <code>dimcli</code>	103
7.4	Adding New Columns to Output	109
7.5	Adding New Commands	111
7.6	Adding a Module in Another Repo	122
8	Service Discovery Using Consul	125
9	Debugging and Development	129
9.1	Determining File System Affects of Running Programs	129
10	Docker Datacenter	135
10.1	Datacenter Walk-thru	135
10.2	Further Information	137
11	License	139

This is the DIMS User Manual (version 0.2.1).

CHAPTER 1

Introduction

Introduction

This chapter introduces ...

CHAPTER 2

Referenced documents

1. Contract HSHQDC-13-C-B0013, “From Local to Global Awareness: A Distributed Incident Management System,” Section C - Statement of Work
2. [DIMS Training Manual v 0.3.0](#)
3. [DIMS Job Descriptions v 2.9.0](#)

Development and Core Tool Policy

This section contains policy statements regarding software development that all developers working on the DIMS project are expected to adhere to.

In order to prevent core tools being used by developers being incompatible, rendering installation instructions buggy and/or causing random failures in a complicated build environment, everyone on the DIMS project **must** use the same core tools, and use the same workflow processes. This will allow controlled updates and provide stability in the tools we are using within the project.

Without some discipline and adherence to documented policies, far too much time ends up being wasted when one person can do something, but another can't, or something runs fine on one system, but fails on another system. In either case, team members get blocked from making forward progress and the project suffers as a result. These policies are not being imposed to stifle anyone's creativity, but to help *everyone* on the team be more productive.

Attention: The requirement to adhere to the policies stated here is partly to keep the project moving forward smoothly, but also to ensure that the software products developed by this project are suitable for public open source release as required by the contract (see `dimssr:opensource` in `dimssr:dimssystemrequirements`) and in conformance with University of Washington policy.

General Software Development Philosophy

This section covers some very high-level philosophical points that DIMS software developers should keep in mind.

There are a huge [List of software development philosophies](#) on Wikipedia. One of the most relevant to the DIMS project, based on a contractual requirement (see `dimssr:agileDevelopment`) is the [Agile Manifesto](#). This manifesto is based on twelve principles:

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers

5. Projects are built around motivated individuals, who should be trusted
 6. Face-to-face conversation is the best form of communication (co-location)
 7. Working software is the principal measure of progress
 8. Sustainable development, able to maintain a constant pace
 9. Continuous attention to technical excellence and good design
 10. Simplicitythe art of maximizing the amount of work not doneis essential
 11. Self-organizing teams
 12. Regular adaptation to changing circumstance
- **Avoid friction** - “Friction” is anything that slows down an otherwise smooth running process. Little things that are broken, missing facts, new programs that you wrote that don’t yet have any documentation, all make it harder for someone to get work done because something causes friction. Everything grinds to a halt until the little roadblock can be removed and then it takes more time to ramp back up to speed.
 - **Take control** - Relying on the default behaviors that are programmed into an open source product that we use within the DIMS project, without fully understanding them, can cause problems. When possible, being explicit about how programs are configured and how they are invoked can make these opaque default behaviors less of a problem.
 - **Make it simple** - It may take a little effort, but being focused on finding a simple solution that can be applied uniformly makes it easier to intergrate a large set of components. The more differences there are the way a subsystem or service is configured on multiple hosts (like DNS, for example) means the behavior is random and unpredictable from one computer system to another, causing friction
 - **Make it work first, then make it better** - Trying to engineer a complete solution to some need can mean delays in getting something working, which delays getting that component integrated with other components. Or worrying about how slow something might be during initial development and trying to optimize the solution before it is even working and tested by someone else. Make it work first, doing something simple, then deal with optimization and a comprehensive feature set later.
 - **Avoid hard coding!!!** - When ever possible, avoid using hard-coded values in programs, configuration files, or other places where a simple change of plans or naming conventions results in having to go find and edit dozens of files. A complete system made up of multiple services and software components that must be replicated as a whole *cannot* possibly be replicated if someone has to hunt down and change hundreds of values in files spread all over the place.
 - **Ansible-ize all the things** - All configuration, package installation, or entity creation on a computer system should be looked at in terms of how it can be automated with Ansible. Whenever you are tempted to run a command to change something, or fire up an editor to set a variable, *put it in Ansible and use Ansible to apply it*. Manual processes are not well documented, are not managed under version control, are not programatically repeatable, and make it harder to scale or replicate a system of systems because they cause *huge* amounts of friction.
 - **Template and version control all configuration** - Adding a new service (e.g., Nginx, or RabbitMQ) that may have several configuration files is not just a one-time task. It will be repeated many times, for testing, for alternate deployments, or when hardware fails or virtual machines get upgraded. Don’t think that cutting corners to get something up and running fast by just hand-configuration is the right way to go, because doing it again will take as much time (or maybe even longer, if someone unfamiliar with the process has to do it the next time). Take the time when adding a new service to learn how it is configured, put all of its configuration files under Ansible control, and use Ansible playbooks or other scripts to do the configuration at deployment time and at runtime.
 - **Done means someone else can do it, not just you.** A program that compiles, but nobody else can run, is not done. A bug that was fixed, but hasn’t been tested by someone other than the person who wrote the code or fixed

the bug, is not done. Something that doesn't have documentation, or test steps that explain how to replicate the results, are not done.

- **You can't grade your own exam** Tickets should not be closed until someone else on the team has been able to validate the results.
- **Document early, document often** - A program that has no documentation, or a process someone learns that has no documentation to record that knowledge and how to use it, doesn't contribute much to moving the project forward. We are a team who mostly works independently, across multiple timezones and on different daily schedules.

Source Code Control

As pointed out by Jeff Knupp in his blog post [Open Sourcing a Python Project the Right Way](#), "git and GitHub have become the de-facto standard for Open Source projects." Just as Knupp's post suggests, the DIMS project has been following the same *git-flow* model described by Vincent Driesen in his [A successful Git branching model](#) blog post, using *Sphinx* and *RST* (see the section [Documenting DIMS Components](#)), and using *continuous integration* via Jenkins (see [Continuous Integration](#)).

Copyright

All source code should include a copyright statement with the year the project started (2013) and the current year, as shown here:

```
#!/usr/bin/env python
#
# Copyright (C) 2013, 2015 University of Washington. All rights reserved.
#
# ...
```

Note: Where possible, include the actual copyright symbol. For example, in Sphinx documents, follow the instructions in Section [Insertion of text using direct substitution](#).

License

All source code repositories shall include the following license statement to accompany the Copyright statement in the previous section.

```
Berkeley Three Clause License
=====

Copyright (c) 2014, 2016 University of Washington. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
```

2. Redistributions **in** binary form must reproduce the above copyright notice, this **list** of conditions **and** the following disclaimer **in** the documentation **and/or** other materials provided **with** the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse **or** promote products derived **from this** software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Developing on a fork from GitHub

In this section, we will go through the steps for using Hub Flow for developing on a branch forked from GitHub, publishing the results back to GitHub for others to share.

For this example, there has already been a fork made on GitHub. Start by cloning it to your local workstation:

```
[dittrich@localhost git (master)]$ git clone https://github.com/uw-dims/sphinx-
↪autobuild.git
Cloning into 'sphinx-autobuild'...
remote: Counting objects: 366, done.
remote: Total 366 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (366/366), 62.23 KiB | 0 bytes/s, done.
Resolving deltas: 100% (180/180), done.
Checking connectivity... done.
[dittrich@localhost git (master)]$ cd sphinx-autobuild/
[dittrich@localhost sphinx-autobuild (develop)]$ git branch -a
* develop
  remotes/origin/HEAD -> origin/develop
  remotes/origin/develop
  remotes/origin/feature/1-arbitrary-watch
  remotes/origin/feature/tests
  remotes/origin/master
[dittrich@localhost sphinx-autobuild (develop)]$ git checkout master
Branch master set up to track remote branch master from origin by rebasing.
Switched to a new branch 'master'
[dittrich@localhost sphinx-autobuild (master)]$ git branch -a
  develop
* master
  remotes/origin/HEAD -> origin/develop
  remotes/origin/develop
  remotes/origin/feature/1-arbitrary-watch
  remotes/origin/feature/tests
  remotes/origin/master
[dittrich@localhost sphinx-autobuild (develop)]$ ls
AUTHORS                                NEWS.rst                                fabfile.py
↪ requirements-testing.txt
```

CONTRIBUTING.rst	README.rst	fabtasks	↵
↵ requirements.txt			
LICENSE	docs	pytest.ini	↵
↵ setup.py			
MANIFEST.in	entry-points.ini	requirements-dev.txt	↵
↵ sphinx_autobuild			

Developing

Developing new features for the DIMS CI Utilities...

Source Code Management with Git

Daily development work on DIMS source code is done using a local server accessed via SSH to `git.prisem.washington.edu`. The public release of DIMS software will be from github.com/uw-dims with public documentation delivered on [ReadTheDocs](#). (DIMS documentation is covered in Section *Documenting DIMS Components*.)

Note: At this point github.com/uw-dims primarily contains forked repositories of the software described in Section *installingtools*.

Team members need to have familiarity with a few general task sets, which are covered in the sections below. These tasks include things like:

- Cloning repositories and initializing them for use of the `hub-flow` Git addon scripts.
- On a daily basis, updating repositories, creating feature or hotfix branches to work on projects, and finishing those branches after testing is complete to merge them back into the `develop` branch.
- Creating new repositories, setting triggers for post-commit actions, and monitoring continuous integration results.
- Keeping up to date with new repositories (or starting fresh with a new development system by cloning all DIMS repositories a new.)

Attention: Every now and then, you may do something with Git and immediately think, “Oh, snap! I did *not* want to do *that...*” :(

There are resource on Dave Dittrich’s home page in the `dittrich:usinggit` section. Two good resources for learning how things work with Git (and how to undo them) are:

- [How to undo \(almost\) anything with Git](#), GitHub blog post by jaw6, June 8, 2015
- [Undo Almost Anything with Git webinar](#), YouTube video by Peter Bell and Michael Smith, February 11, 2014

Foundational Git Resources

- [Yan Pritzker's Git Workflows book](#)
- [The Thing About Git](#)
- [Commit Often, Perfect Later, Publish Once: Git Best Practices](#)
- [Git Tips](#)
- [git-flow](#) utilities to follow Vincent Dreisen branching workflow
- [HubFlow](#) (GitFlow for GitHub)

The need for policy and discipline

Git is a great tool for source management, but can be a little tricky to use when there is a team of programmers all using Git in slightly different ways. Bad habits are easy to form, like the short-cut of working on the `develop` branch in a multi-branch workflow.

Figure *Vincent Driessen Git branching model* comes from Vincent Driessen's "[A successful Git branching model](#)". The DIMS project is following this model as best we can to maintain consistency in how we create and use branches. The general policy is to derive branch names from Jira tickets, in order to keep information about why the branch exists, who is responsible for working on it, and what is supposed to be done on the branch, in a system that can track progress and prioritization of activities within sprints.

Because public release of source code will be through GitHub, the `hubflow` tool was chosen for use within the project. Take a moment to read through the following Gist (original source: [bevanhunt/hubflow_workflow](#)), just to get an overview of `hubflow` concepts. This Gist provides an overview of `hubflow` branch concepts and some other things about Git that are good to keep in mind, but this is *not* the totality of information in this guide about using `hubflow` (keep reading further down for more DIMS-specific examples of using `hubflow` commands).

```
Git Hubflow Workflow:
```

```
Sync Branch:
```

```
git hf update - this will update master and develop and sync remote branches_
↳withlocal ones (be sure not to put commits into develop or master as it will push_
↳these up)
git hf push - this will push your commits in your local branch to the matching remote_
↳branch
git hf pull - this will pull the remote commits into your local branch (don't use if_
↳the remote branch has been rebased - use git pull origin "your-branch" instead)
```

```
Feature Branch:
```

```
git hf feature start "my-feature" - this will create a feature branch on origin and_
↳local will be based off the latest develop branch (make sure to git hf update_
↳before or you will get an error if local develop and remote develop have diverged)
git hf feature finish "my-feature" - this will delete the local and remote branches_
↳(only do this after a PR has been merged)
git hf feature cancel -f "my-feature" - this will delete the local and remote_
↳branches (only do this if the feature branch was created in error)
git hf feature checkout "my-feature" - this will checkout the feature branch
```

```
Hotfix Branch:
```

```
git hf hotfix start "release-version" - this will create a hotfix branch on origin_
↳and local will be based off the latest develop branch (make sure to git hf update_
↳before or you get an error if local develop and remote develop have diverged)
```

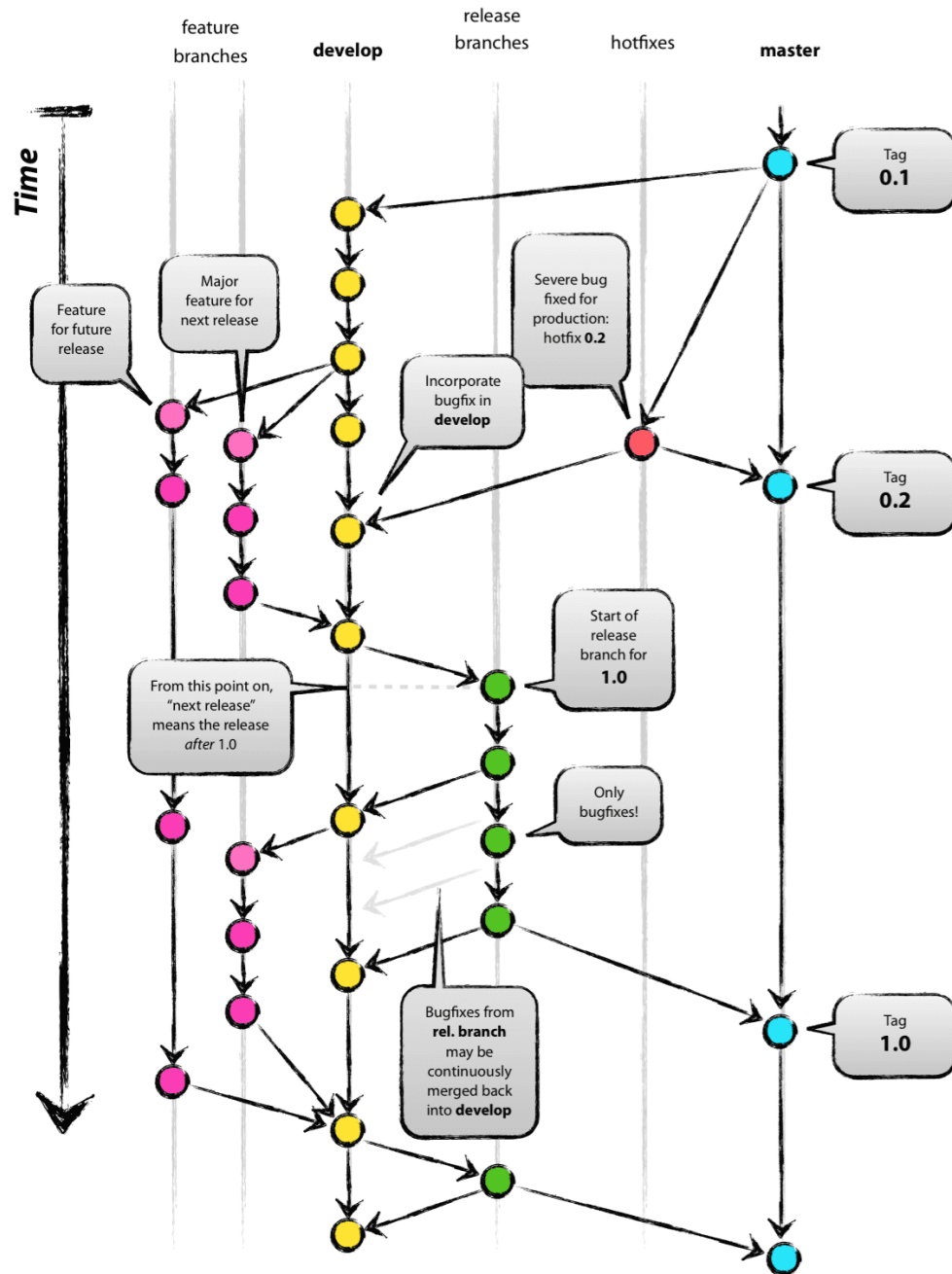


Fig. 4.1: Vincent Driessen Git branching model

```
git hf hotfix finish "release-version" - this will delete the local and remote_
↳branches and merge the commits of the hotfix branch into master and develop_
↳branches - it will also create a release tag that matches the release version on_
↳master
git hf hotfix cancel -f "release-version" - this will delete the remote and local_
↳branch (only do this if the hotfix was created in error)
git checkout hotfix/"release-version" - this will checkout the hotfix branch (make_
↳sure to git hf update first)
```

Release Branch:

```
git hf release start "release-version" - this will create a release branch on origin_
↳and local will be based off the latest develop branch (make sure to git hf update_
↳before or you get an error if local develop and remote develop have diverged)
git hf release finish "release-version" - this will delete the local and remote_
↳branches and merge the commits of the release branch both into develop and master -
↳it will also create a release tag that matches the release version on master
git hf release cancel -f "release-version" - this will delete the local and remote_
↳branch (only do this if the release was created in error)
git checkout release/"release-version" - this will checkout the release branch (make_
↳sure to git hf update first)
```

Preparing a PR:

- put the Aha! Ticket # in PR title with a description
- assign to the proper reviewer
- don't squash the commits until after reviewed
- after review - squash the commits

Squashing Commits:

- checkout the branch you want to squash
- git merge-base "my-branch" develop (returns merge-base-hash)
- git rebase -i "merge-base-hash"
- change all commit types to "squash" from "pick" in the text file (except first) &_
↳save file
- if you get a no-op message in the text file and still have multiple commits then_
↳use the command git rebase -i (without the hash)
- fix any merge conflicts
- you should have one commit
- force update your remote branch: git push origin "my-branch" -f

Resolving merge conflicts with the develop branch that are not squashing related_

- ↳(generally on PRs - auto-merge will show as disabled):
- git hf update
- git rebase develop (while in your branch)
- resolve any merge conflicts

Rules to remember:

- don't ever git merge branches together manually (should never run command - git_
↳merge)
- squash only after review and before merging PR into develop

Note: There is a large body of references on Git that are constantly being updated in the [Software Development>Git](#) section of Dave Dittrich's web page.

Caution: Mac OS X (by default) uses an HFS file system *with case sensitivity*. Unlike Ubuntu and other Linux/Unix distributions using case-sensitive file systems like ext2, reiserfs, etc., the default OS X file system does not care if you name a file `THISFILE` or `ThisFile` or `thisfile`. All of those are the same file name. This can cause problems when you use Git to share a source repository between computers running OS X, Windows, and/or Linux. See [Git on Mac OS X: Don't ignore case!](#) and [How do I commit case-sensitive only filename changes in Git?](#). A solution for Mac OS X, posted in [Case sensitivity in Git](#), is documented in Section `macosxcasesensitive`.

Global Git Configuration

As we learn about best practices, the following set of global configuration settings will be updated. Refer back to this page, or look in the `dims-git` repo, for the latest configuration examples.

The following are user-specific settings that you should alter for your own account and preferences of editor/merge method:

```
$ git config --global user.name "Dave Dittrich"
$ git config --global user.email "dittrich@u.washington.edu"
$ git config --global merge.tool vimdiff
$ git config --global core.editor vim
```

The following are general and can be applied to anyone's configuration (included here without a prompt so you can cut/paste to a command line):

```
git config --global push.default tracking
git config --global core.excludesfile ~/.gitignore_global
git config --global core.autocrlf false
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
git config --global color.interactive auto
git config --global color.ui auto
git config --global branch.autosetuprebase always
```

The following are convenience aliases that help with certain tasks:

```
git config --global alias.find 'log --color -p -S'
git config --global alias.stat 'status -s'
git config --global alias.unstage "reset HEAD --"
git config --global alias.uncommit "reset --soft HEAD^"
git config --global alias.gr 'log --full-history --decorate=short --all --color --
↳graph'
git config --global alias.lg 'log --oneline --decorate=short --abbrev-commit --all --
↳color --graph'
git config --global alias.log1 'log --oneline --decorate=short'
```

Daily tasks with Git

This section covers regular tasks that are performed to work with source code using Git. This section assumes you are using the `hub` `flow` tool described in Section `installingtools`.

Warning: These tools are being installed in the `dimsenv` Python virtual environment to make it easier for everyone on the team to access them and to stay up to date with instructions in this document. If you have *any* problems, file a [Jira](#) ticket or talk to Dave immediately upon encountering a problem. Do not let yourself get blocked on something and block everyone else as a result!

Updating local repos

The most common task you need to do is keep your local Git repos up to date with the code that others have pushed to remote repositories for sharing. With several dozen individual Git repos, keeping your system up to date with all of these frequently changing repos using `git` commands alone is difficult.

To make things easier, helper programs like the `hubflow` scripts and `mr` can be used, but even those programs have their limits.

The preferred method of updating the larger set of DIMS Git repos is to use `dims.git.syncrepos`, which in turn calls `hubflow` via `mr` as part of its processing. This convenience script (described in Section [Updating with `dims.git.syncrepos`](#)) works on many repos at once, saving time and effort.

You should still learn how `hubflow` and `mr` work, since you will need to use them to update individual Git repos when you are working within those repos, so we will start with those tools.

Updating using `hubflow`

The following command ensures that a local repo you are working on is up to date:

Note: The list of actions that are performed is provided at the end of the command output. This will remind you of what all is happening under the hood of Hub Flow and is well worth taking a few seconds of your attention.

```
(dimsenv)[dittrich@localhost ansible-playbooks (develop)]$ git hf update
Fetching origin
remote: Counting objects: 187, done.
remote: Compressing objects: 100% (143/143), done.
remote: Total 165 (delta 56), reused 1 (delta 0)
Receiving objects: 100% (165/165), 31.78 KiB | 0 bytes/s, done.
Resolving deltas: 100% (56/56), completed with 13 local objects.
From git.prisem.washington.edu:/opt/git/ansible-playbooks
   001ba47..0e12ec3  develop    -> origin/develop
   * [new branch]      feature/dims-334 -> origin/feature/dims-334
Updating 001ba47..0e12ec3
Fast-forward
 docs/source/conf.py | 2 +-
 roles/dims-ci-utils-deploy/tasks/main.yml | 5 +++++
 2 files changed, 6 insertions(+), 1 deletion(-)

Summary of actions:
- Any changes to branches at origin have been downloaded to your local repository
- Any branches that have been deleted at origin have also been deleted from your
  ↪ local repository
- Any changes from origin/master have been merged into branch 'master'
- Any changes from origin/develop have been merged into branch 'develop'
- Any resolved merge conflicts have been pushed back to origin
- You are now on branch 'develop'
```

If a branch existed on the remote repo (e.g., the `feature/eliot` branch used in testing), it would be deleted:

```
[dittrich@localhost dims-asbuilt (develop)]$ git branch -a
* develop
  master
  remotes/origin/develop
  remotes/origin/feature/eliot
  remotes/origin/master
[dittrich@localhost dims-asbuilt (develop)]$ git hf update
Fetching origin
From git.prisem.washington.edu:/opt/git/dims-asbuilt
 x [deleted]          (none)      -> origin/feature/eliot

Summary of actions:
- Any changes to branches at origin have been downloaded to your local repository
- Any branches that have been deleted at origin have also been deleted from your
  ↳ local repository
- Any changes from origin/master have been merged into branch 'master'
- Any changes from origin/develop have been merged into branch 'develop'
- Any resolved merge conflicts have been pushed back to origin
- You are now on branch 'develop'
[dittrich@localhost dims-asbuilt (develop)]$ git branch -a
* develop
  master
  remotes/origin/develop
  remotes/origin/master
```

While using `git hf update` && `git hf pull` seems like it is simple enough, the DIMS project has several dozen repos, many of which are inter-related. Keeping them all up to date is not simple, and because of this developers often get far out of sync with the rest of the team.

Updating using the `mr` command

A useful tool for managing multiple Git repositories and keeping them in sync with the master branches is to use the program `mr`.

`mr` uses a configuration file that can be added to using `mr register` within a repo, or by editing/writing the `.mrconfig` file directly.

Attention: These instructions assume the reader is *not already using* `mr` on a regular basis. Additionally, all DIMS Git repos are assumed to be segregated into their own directory tree apart from any other Git repos that the developer may be using.

This assumption allows for use of a `.mrconfig` file specifically for just DIMS source code that can be overwritten entirely with DIMS-specific settings.

Cloning all of the DIMS source repos at once, or getting the contents of what should be an up-to-date `.mrconfig` file, is covered in the Section [Cloning multiple repos from git.prisem.washington.edu](#).

After all repos have been cloned, they can be kept up to date on a daily basis. Start your work session with the following commands:

```
$ cd $GIT
$ mr update
```

Caution: If you do not update a repo before attempting to `git hf push` or `git hf update` with committed changes, Git will do a `pull` and potentially you will end up with at best a merge, and at worst a merge conflict that you must resolve before the push can complete. If you are not comfortable handling a merge conflict, talk to another team member to get help.

Updating with `dims.git.syncrepos`

A script that combines several of the above steps into one single command is `dims.git.synrepos`.

```
[dimsenv] dittrich@dimsdemo1:~ () $ dims.git.syncrepos --version
dims.git.syncrepos version 1.6.97
```

In the example here, highlighted lines show where repos are *dirty* (Repo[9], Repo[13], and Repo[33]), meaning they have tracked files that are not committed yet and cannot be updated, *clean* and requiring updates from the remote repo (Repo[12]), and new repositories from the remote server (Repo[28] and Repo[30]) that are being cloned and initialized for use with *hub-flow* tools. At the end, `dims.git.syncrepos` reports how many repos were updated out of the available repos on the remote server, how many new repos it added, and/or how many repos could not be updated because they are dirty. Lastly, it reports how long it took (so you can be aware of how long you have to go get coffee after starting a sync.)

```
1 [dimsenv] dittrich@dimsdemo1:~ () $ dims.git.syncrepos
2 [+++] Found 46 available repos at git@git.prisem.washington.edu
3 [+++] Repo[1] "/home/dittrich/dims/git/ansible-inventory" clean:
4 [+++] Repo[2] "/home/dittrich/dims/git/ansible-playbooks" clean:
5 [+++] Repo[3] "/home/dittrich/dims/git/cif-client" clean:
6 [+++] Repo[4] "/home/dittrich/dims/git/cif-java" clean:
7 [+++] Repo[5] "/home/dittrich/dims/git/configs" clean:
8 [+++] Repo[6] "/home/dittrich/dims/git/dims" clean:
9 [+++] Repo[7] "/home/dittrich/dims/git/dims-ad" clean:
10 [+++] Repo[8] "/home/dittrich/dims/git/dims-asbuilt" clean:
11 [---] Repo[9] "/home/dittrich/dims/git/dims-ci-utils" is dirty:
12 ?? dims/diffs.1
13 ?? dims/manifest.dat
14 ?? ubuntu-14.04.2/ubuntu-14.04.3-install.dd.bz2
15 4bb5516 (feature/dims-406) Merge branch 'develop' into feature/dims-406
16
17 [+++] Repo[10] "/home/dittrich/dims/git/dims-dashboard" clean:
18 [+++] Repo[11] "/home/dittrich/dims/git/dims-db-recovery" clean:
19 [+++] Repo[12] "/home/dittrich/dims/git/dims-devguide" clean:
20 remote: Counting objects: 29, done.
21 remote: Compressing objects: 100% (22/22), done.
22 remote: Total 22 (delta 13), reused 0 (delta 0)
23 Unpacking objects: 100% (22/22), done.
24 From git.prisem.washington.edu:/opt/git/dims-devguide
25 daffa68..4b2462b develop -> origin/develop
26 Updating daffa68..4b2462b
27 Fast-forward
28 .bumpversion.cfg | 2 +-
29 docs/source/conf.py | 4 +--
30 docs/source/deployconfigure.rst | 2 +-
31 docs/source/referenceddocs.rst | 13 ++++++++
32 4 files changed, 17 insertions(+), 4 deletions(-)
33 [---] Repo[13] "/home/dittrich/dims/git/dims-dockerfiles" is dirty:
34 8a47fca (HEAD -> develop) Bump version: 1.1.11 -> 1.1.12
35
```



```

36  [+++] Repo[14] "/home/dittrich/dims/git/dims-dsdd" clean:
37  [+++] Repo[15] "/home/dittrich/dims/git/dims-jds" clean:
38  [+++] Repo[16] "/home/dittrich/dims/git/dims-keys" clean:
39  [+++] Repo[17] "/home/dittrich/dims/git/dims-ocd" clean:
40  [+++] Repo[18] "/home/dittrich/dims/git/dims-packer" clean:
41  [+++] Repo[19] "/home/dittrich/dims/git/dims-sample-data" clean:
42  [+++] Repo[20] "/home/dittrich/dims/git/dims-sr" clean:
43  [+++] Repo[21] "/home/dittrich/dims/git/dims-supervisor" clean:
44  [+++] Repo[22] "/home/dittrich/dims/git/dims-svd" clean:
45  [+++] Repo[23] "/home/dittrich/dims/git/dimssysconfig" clean:
46  [+++] Repo[24] "/home/dittrich/dims/git/dims-tp" clean:
47  [+++] Repo[25] "/home/dittrich/dims/git/dims-tr" clean:
48  [+++] Repo[26] "/home/dittrich/dims/git/dims-vagrant" clean:
49  [+++] Repo[27] "/home/dittrich/dims/git/ELK" clean:
50  [+++] Adding Repo[28] fuse4j to /home/dittrich/dims/.mrconfig and checking it out.
51  mr checkout: /home/dittrich/dims/git/fuse4j
52  Cloning into 'fuse4j'...
53  remote: Counting objects: 523, done.
54  remote: Compressing objects: 100% (240/240), done.
55  remote: Total 523 (delta 186), reused 523 (delta 186)
56  Receiving objects: 100% (523/523), 180.86 KiB | 0 bytes/s, done.
57  Resolving deltas: 100% (186/186), done.
58  Checking connectivity... done.
59  Using default branch names.
60
61  Which branch should be used for tracking production releases?
62  - master
63  Branch name for production releases: [master]
64  Branch name for "next release" development: [develop]
65
66  How to name your supporting branch prefixes?
67  Feature branches? [feature/]
68  Release branches? [release/]
69  Hotfix branches? [hotfix/]
70  Support branches? [support/]
71  Version tag prefix? []
72
73  mr checkout: finished (1 ok; 43 skipped)
74  [+++] Repo[29] "/home/dittrich/dims/git/ipgrep" clean:
75  [+++] Adding Repo[30] java-native-loader to /home/dittrich/dims/.mrconfig and
↪checking it out.
76  mr checkout: /home/dittrich/dims/git/java-native-loader
77  Cloning into 'java-native-loader'...
78  remote: Counting objects: 329, done.
79  remote: Compressing objects: 100% (143/143), done.
80  remote: Total 329 (delta 62), reused 329 (delta 62)
81  Receiving objects: 100% (329/329), 178.36 KiB | 0 bytes/s, done.
82  Resolving deltas: 100% (62/62), done.
83  Checking connectivity... done.
84  Using default branch names.
85
86  Which branch should be used for tracking production releases?
87  - master
88  Branch name for production releases: [master]
89  Branch name for "next release" development: [develop]
90
91  How to name your supporting branch prefixes?
92  Feature branches? [feature/]

```

```
93 Release branches? [release/]
94 Hotfix branches? [hotfix/]
95 Support branches? [support/]
96 Version tag prefix? []
97
98 mr checkout: finished (1 ok; 44 skipped)
99 [++] Repo[31] "/home/dittrich/dims/git/java-stix-v1.1.1" clean:
100 [++] Repo[32] "/home/dittrich/dims/git/mal4s" clean:
101 [---] Repo[33] "/home/dittrich/dims/git/MozDef" is dirty:
102     M docker/Dockerfile
103     M docker/Makefile
104
105 [++] Repo[34] "/home/dittrich/dims/git/ops-trust-openid" clean:
106 [++] Repo[35] "/home/dittrich/dims/git/ops-trust-portal" clean:
107 [++] Repo[36] "/home/dittrich/dims/git/poster-deck-2014-noflow" clean:
108 [++] Repo[37] "/home/dittrich/dims/git/prisem" clean:
109 [++] Repo[38] "/home/dittrich/dims/git/prisem-replacement" clean:
110 [++] Repo[39] "/home/dittrich/dims/git/pygraph" clean:
111 [++] Repo[40] "/home/dittrich/dims/git/rwfind" clean:
112 [---] Repo[41] "/home/dittrich/dims/git/sphinx-autobuild" is clean:
113 [++] Repo[42] "/home/dittrich/dims/git/stix-java" clean:
114 [++] Repo[43] "/home/dittrich/dims/git/ticketing-redis" clean:
115 [++] Repo[44] "/home/dittrich/dims/git/tsk4j" clean:
116 [++] Repo[45] "/home/dittrich/dims/git/tupelo" clean:
117 [++] Repo[46] "/home/dittrich/dims/git/umich-botnets" clean:
118 [++] Updated 40 of 46 available repos.
119 [++] Summary of actions for repos that were updated:
120 - Any changes to branches at origin have been downloaded to your local repository
121 - Any branches that have been deleted at origin have also been deleted from your ↵
↵ local repository
122 - Any changes from origin/master have been merged into branch 'master'
123 - Any changes from origin/develop have been merged into branch 'develop'
124 - Any resolved merge conflicts have been pushed back to origin
125 [++] Added 3 new repos: fuse4j java-native-loader tsk4j
126 [++] Could not update 3 repos: dims-ci-utils dims-dockerfiles MozDef
127 [++] Updating repos took 00:04:12
```

Managing Version Numbers

The DIMS project uses the Python program `bumpversion` to update version numbers in Git repositories, following [PEP 440 – Version Identification and Dependency Specification](#). You can learn how `bumpversion` works from these resources:

- [bumpversion screencast](#) showing `bumpversion` in action
- [A Python Versioning Workflow With Bumpversion](#)

Note: You can find examples of using `bumpversion` (including its configuration file `.bumpversion.cfg` and how it is used to manage version numbers in files) in this document in Sections [Creating a new documentation-only repo](#) and [Cherry-picking a commit from one branch to another](#).

The program `bumpversion` is included in the Python virtual environment `dimsenv` that is created for use in DIMS development.

```
[dimsenv] dittrich@27b:~/git/homepage (develop*) $ which bumpversion
/Users/dittrich/dims/envs/dimsenv/bin/bumpversion
```

Caution: Because you must be in the same directory as the `.bumpversion.cfg` file when you invoke `bumpversion`, it is sometimes problematic when using it to work in a sub-directory one or more levels below the configuration file. You may see example command lines like `(cd ../; bumpversion patch)` that use sub-shells to temporarily change to the right directory, do the `bumpversion patch`, then exit leaving you in the same directory where you are editing files. That is a little more work than is desirable, but doing a bunch of `cd ../, bumpversion patch, cd backagain` is even more work.

To make it easier to increment version numbers, a helper script `dims.bumpversion` is available as well:

```
[dimsenv] dittrich@27b:~/git/homepage (develop*) $ which dims.bumpversion
/Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion
[dimsenv] dittrich@27b:~/git/homepage (develop*) $ dims.bumpversion --help
Usage:
/Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion [options] [args]

Use "/Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion --help" to see options.
Use "/Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion --usage" to see help on
↳ "bumpversion" itself.

/Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion -- [bumpversion_options]↳
↳ [bumpversion_args]

Follow this second usage example and put -- before any bumpversion
options and arguments to pass them on bumpversion (rather than
process them as though they were /Users/dittrich/dims/envs/dimsenv/bin/dims.
↳ bumpversion arguments.) After
all, /Users/dittrich/dims/envs/dimsenv/bin/dims.bumpversion is just a shell wrapping↳
↳ bumpversion.

Options:
  -h, --help      show this help message and exit
  -d, --debug     Enable debugging
  -u, --usage     Print usage information.
  -v, --verbose   Be verbose (on stdout) about what is happening.
```

The default when you just invoke `dims.bumpversion` is to do `bumpversion patch`, the most frequent version increment. To use a different increment, just add it as an argument on the command line (e.g., `dims.bumpversion minor`).

Here is an example of how this section edit was done, showing the version number increment in the workflow:

```
[dimsenv] dittrich@localhost:~/dims/git/dims-devguide/docs (develop*) $ git add↳
↳ source/sourcemanagement.rst
[dimsenv] dittrich@localhost:~/dims/git/dims-devguide/docs (develop*) $ git stat
M docs/source/sourcemanagement.rst
[dimsenv] dittrich@localhost:~/dims/git/dims-devguide/docs (develop*) $ git commit -
↳ m "Add subsection on version numbers and bumpversion/dims.bumpversion"
[develop b433bae] Add subsection on version numbers and bumpversion/dims.bumpversion
1 file changed, 92 insertions(+)
[dimsenv] dittrich@localhost:~/dims/git/dims-devguide/docs (develop*) $ dims.
↳ bumpversion
[dimsenv] dittrich@localhost:~/dims/git/dims-devguide/docs (develop*) $ git hf push
```

```

Fetching origin
Already up-to-date.
Counting objects: 11, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 2.53 KiB | 0 bytes/s, done.
Total 11 (delta 7), reused 0 (delta 0)
remote: Running post-receive hook: Thu Oct 22 22:31:50 PDT 2015
remote: [+++] post-receive-06jenkinsalldocs started
remote: [+++] REPONAME=dims-devguide
remote: [+++] BRANCH=develop
remote: [+++] newrev=00727d53dbc8130cddbde35be80f1f4c2d2ee7fa
remote: [+++] oldrev=e8e7d4db40dd852a044525fdfbada1fe80d81739
remote: [+++] Branch was updated.
remote: [+++] This repo has a documentation directory.
remote:  % Total      % Received % Xferd  Average Speed   Time    Time       Time  └─
↪Current
remote:
remote:      Dload  Upload  Total  Spent    Left  Speed
remote: 100      79    0      0  100      79      0   2613 --:--:-- --:--:-- --:--:-- └─
↪3761
remote:  % Total      % Received % Xferd  Average Speed   Time    Time       Time  └─
↪Current
remote:
remote:      Dload  Upload  Total  Spent    Left  Speed
remote: 100      78    0      0  100      78      0   2524 --:--:-- --:--:-- --:--:-- └─
↪3250
remote: [+++] post-receive-06jenkinsalldocs finished
To git@git.prisem.washington.edu:/opt/git/dims-devguide.git
   e8e7d4d..00727d5  develop -> develop

Summary of actions:
- The remote branch 'origin/develop' was updated with your changes

```

Initializing a repo for hub-flow

Every time you clone a new DIMS repo, it must be initialized with hub-flow so that hub-flow commands work properly. Initialize your repo this way:

```

(dimsenv)[dittrich@localhost git]$ git clone git@git.prisem.washington.edu:/opt/git/
↪dims-ad.git
Cloning into 'dims-ad'...
remote: Counting objects: 236, done.
remote: Compressing objects: 100% (155/155), done.
remote: Total 236 (delta 117), reused 159 (delta 76)
Receiving objects: 100% (236/236), 26.20 MiB | 5.89 MiB/s, done.
Resolving deltas: 100% (117/117), done.
Checking connectivity... done.
(dimsenv)[dittrich@localhost git]$ cd dims-ad
(dimsenv)[dittrich@localhost dims-ad (master)]$ git hf init
Using default branch names.

Which branch should be used for tracking production releases?
- master
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?

```

```
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
```

After initializing hub-flow, there will be two new sections in your `.git/config` file starting with hubflow:

```
(dimsenv)[dittrich@localhost dims-ad (develop)]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
    precomposeunicode = true
[remote "origin"]
    url = git@git.prisem.washington.edu:/opt/git/dims-ad.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
    rebase = true
[hubflow "branch"]
    master = master
    develop = develop
[branch "develop"]
    remote = origin
    merge = refs/heads/develop
    rebase = true
[hubflow "prefix"]
    feature = feature/
    release = release/
    hotfix = hotfix/
    support = support/
    versiontag =
```

Note: A possible test for inclusion in the `dims-ci-utils` test suite would be to check for the existence of the hubflow "branch" and hubflow "prefix" sections.

These are automatically created when repos are checked out using the `dims.git.syncrepos` script and/or methods involving `mr` described in the following sections.

Infrequent tasks with Git

Cloning multiple repos from `git.prisem.washington.edu`

There are several dozen repositories on `git.prisem.washington.edu` that contain DIMS-generated code, configuration files, and/or documentation, but also local copies of Git repositories from other sources (some with DIMS-related customizations).

To get a list of all repositories on `git.prisem.washington.edu`, use the Git shell command `list`:

```
[dittrich@localhost ~]$ ssh git@git.prisem.washington.edu list
prisem-replacement.git
ELK.git
cif-java.git
cif-client.git
dims-ad.git
supervisor.git
dims-tr.git
lemonldap-ng.git
pygraph.git
parsons-docker.git
configs.git
poster-deck-2014-noflow.git
dims-keys.git
dims.git
dims-tp.git
ops-trust-portal.git
dimssysconfig.git
dims-dockerfiles.git
stix-java.git
ansible-playbooks.git
dims-dashboard.git
mal4s.git
dims-ocd.git
sphinx-autobuild.git
dims-devguide.git
dims-asbuilt.git
ticketing-redis.git
dims-sr.git
prisem.git
umich-botnets.git
dims-dsdd.git
dims-sample-data.git
packer.git
java-stix-v1.1.1.git
vagrant.git
dims-jds.git
ansible-inventory.git
ops-trust-openid.git
dims-coreos-vagrant.git
configstest.git
poster-deck-2014.git
rwfind.git
dims-ci-utils.git
ipgrep.git
tupelo.git
dims-opst-portal.git
lemonldap-dims.git
MozDef.git
tsk4j.git
dims-svd.git
```

To clone all of these repositories in a single step, there is another Git shell command `mrconfig` that returns the contents of a `.mrconfig` file (see `man mr` for more information).

Caution: To use a `.mrconfig` file in an arbitrary directory, you will need to add the directory path to this file to the `~/.mrtrust` file. In this example, we will clone repos into `~/dims/git` by placing the `.mrconfig` file in the `~/dims` directory.

```
[dittrich@localhost dims]$ cat ~/.mrtrust
/Users/dittrich/dims/.mrconfig
/Users/dittrich/git/.mrconfig
```

If you are building a documentation set (i.e., a limited set of documentation-only repositories that are cross-linked using the `intersphinx` extension to Sphinx as described in Section [Cross-referencing between documents with the `sphinx.ext.intersphinx` extension](#).

```
[dittrich@localhost ~]$ cd ~/dims
[dittrich@localhost dims]$ ssh git@git.prisem.washington.edu mrconfig dims-ad dims-
↪sr dims-ocd
[git/dims-ad]
checkout = git clone 'git@git.prisem.washington.edu:/opt/git/dims-ad.git' 'dims-ad' &
↪&
    (cd dims-ad; git hf init)
show = git remote show origin
update = git hf update
pull = git hf update &&
    git hf pull
stat = git status -s

[git/dims-sr]
checkout = git clone 'git@git.prisem.washington.edu:/opt/git/dims-sr.git' 'dims-sr' &
↪&
    (cd dims-sr; git hf init)
show = git remote show origin
update = git hf update
pull = git hf update &&
    git hf pull
stat = git status -s

[git/dims-ocd]
checkout = git clone 'git@git.prisem.washington.edu:/opt/git/dims-ocd.git' 'dims-ocd'
↪' &&
    (cd dims-ocd; git hf init)
show = git remote show origin
update = git hf update
pull = git hf update &&
    git hf pull
stat = git status -s
[dittrich@localhost dims]$ ssh git@git.prisem.washington.edu mrconfig dims-ad dims-
↪sr dims-ocd > .mrconfig
[dittrich@localhost dims]$ mr checkout
mr checkout: /Users/dittrich/dims/git/dims-ad
Cloning into 'dims-ad'...
remote: Counting objects: 518, done.
remote: Compressing objects: 100% (437/437), done.
remote: Total 518 (delta 308), reused 155 (delta 76)
Receiving objects: 100% (518/518), 27.88 MiB | 5.88 MiB/s, done.
Resolving deltas: 100% (308/308), done.
Checking connectivity... done.
Using default branch names.
```

```
Which branch should be used for tracking production releases?
```

```
- master
```

```
Branch name for production releases: [master]
```

```
Branch name for "next release" development: [develop]
```

```
How to name your supporting branch prefixes?
```

```
Feature branches? [feature/]
```

```
Release branches? [release/]
```

```
Hotfix branches? [hotfix/]
```

```
Support branches? [support/]
```

```
Version tag prefix? []
```

```
mr checkout: /Users/dittrich/dims/git/dims-ocd
```

```
Cloning into 'dims-ocd'...
```

```
remote: Counting objects: 474, done.
```

```
remote: Compressing objects: 100% (472/472), done.
```

```
remote: Total 474 (delta 288), reused 0 (delta 0)
```

```
Receiving objects: 100% (474/474), 14.51 MiB | 4.26 MiB/s, done.
```

```
Resolving deltas: 100% (288/288), done.
```

```
Checking connectivity... done.
```

```
Using default branch names.
```

```
Which branch should be used for tracking production releases?
```

```
- master
```

```
Branch name for production releases: [master]
```

```
Branch name for "next release" development: [develop]
```

```
How to name your supporting branch prefixes?
```

```
Feature branches? [feature/]
```

```
Release branches? [release/]
```

```
Hotfix branches? [hotfix/]
```

```
Support branches? [support/]
```

```
Version tag prefix? []
```

```
mr checkout: /Users/dittrich/dims/git/dims-sr
```

```
Cloning into 'dims-sr'...
```

```
remote: Counting objects: 450, done.
```

```
remote: Compressing objects: 100% (445/445), done.
```

```
remote: Total 450 (delta 285), reused 0 (delta 0)
```

```
Receiving objects: 100% (450/450), 498.20 KiB | 0 bytes/s, done.
```

```
Resolving deltas: 100% (285/285), done.
```

```
Checking connectivity... done.
```

```
Using default branch names.
```

```
Which branch should be used for tracking production releases?
```

```
- master
```

```
Branch name for production releases: [master]
```

```
Branch name for "next release" development: [develop]
```

```
How to name your supporting branch prefixes?
```

```
Feature branches? [feature/]
```

```
Release branches? [release/]
```

```
Hotfix branches? [hotfix/]
```

```
Support branches? [support/]
```

```
Version tag prefix? []
```

```
mr checkout: finished (3 ok)
```

```
[dittrich@localhost dims]$ mr stat
```



```

mr stat: /Users/dittrich/tmp/dims/git/dims-ad

mr stat: /Users/dittrich/tmp/dims/git/dims-ocd

mr stat: /Users/dittrich/tmp/dims/git/dims-sr

mr stat: finished (3 ok)

```

Note: The example just shown uses only three repos. If you do not specify any repo names on the `mrconfig` Git shell command, it will return the settings for all 50+ DIMS repos. You can then clone the entire set of DIMS repositories with the same `mr checkout` command, and update all of them at once with `mr update`.

Adding a newly-created repository

Until the `dims.git.syncrepos` script has a new feature added to it to detect when a new repo exists on `git.prisem.washington.edu` that does not have a local repo associated with it, you must do this yourself.

When someone uses the `newrepo` script to create a new repo on `git.prisem.washington.edu`, you will need to get new `.mrconfig` settings for that repo in order for `dims.git.syncrepo` to synchronize it. If you have your `$GIT` environment variable pointing to a directory that *only* has DIMS Git repos in it, you just need to create an updated `.mrconfig` file.

Note: It is safest to get a new copy of the `.mrconfig` file contents and save them to a temporary file that you can compare with the current file to ensure you are getting just what you expect, and only then over-writing the `.mrconfig` file with the new contents. The steps are shown here:

```

[dittrich@localhost ~]$ cd $GIT/..
[dittrich@localhost dims]$ ssh git@git.prisem.washington.edu mrconfig > .mrconfig.new
[dittrich@localhost dims]$ diff .mrconfig .mrconfig.new
324a325,333
> [git/dims-db-recovery]
> checkout = git clone 'git@git.prisem.washington.edu:/opt/git/dims-db-recovery.git'
↪ 'dims-db-recovery' &&
>   (cd dims-db-recovery; git hf init)
> show = git remote show origin
> update = git hf update
> pull = git hf update &&
>   git hf pull
> stat = git status -s
>
[dittrich@localhost dims]$ mv .mrconfig.new .mrconfig
[dittrich@27b dims]$ mr checkout
mr checkout: /Users/dittrich/dims/git/dims-db-recovery
Cloning into 'dims-db-recovery'...
remote: Counting objects: 351, done.
remote: Compressing objects: 100% (254/254), done.
remote: Total 351 (delta 63), reused 350 (delta 63)
Receiving objects: 100% (351/351), 7.60 MiB | 5.62 MiB/s, done.
Resolving deltas: 100% (63/63), done.
Checking connectivity... done.
Using default branch names.

```

```
Which branch should be used for tracking production releases?
```

```
- master
```

```
Branch name for production releases: [master]
```

```
Branch name for "next release" development: [develop]
```

```
How to name your supporting branch prefixes?
```

```
Feature branches? [feature/]
```

```
Release branches? [release/]
```

```
Hotfix branches? [hotfix/]
```

```
Support branches? [support/]
```

```
Version tag prefix? []
```

```
mr checkout: finished (1 ok; 43 skipped)
```

Creating Git repositories

As discussed in the introduction to this section, DIMS software will be hosted on both a local server `git.prisem.washington.edu` and from github.com/uw-dims. This section covers creation of new repositories on both systems.

Creating repositories on GitHub

Setting up remote Git repositories on `git.prisem.washington.edu`

Before a repository can be shared between DIMS team members, a remote repository must be set up on `git.prisem.washington.edu` for sharing. The following is an example session creating a new repository named `dims-ocd` for *operational concept description* (a.k.a., *Concept of Operations*).

```
[dittrich@localhost ~]$ slogin git.prisem.washington.edu
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.13.0-43-generic x86_64)
[ ... ]
Last login: Sun Jan 11 12:04:36 2015 from lancaster.prisem.washington.edu
dittrich@jira:~$ sudo su - gituser
[sudo] password for dittrich:
git@jira:~$ cd /opt/git
git@jira:/opt/git$ newrepo dims-ocd.git
Initialized empty Git repository in /opt/git/dims-ocd.git/
git@jira:/opt/git$ echo "DIMS Operational Concept Description" > dims-ocd.git/
↪description
git@jira:/opt/git$ tree dims-ocd.git
dims-ocd.git
+- branches
+- config
+- description
+- HEAD
+- hooks
|   +- post-receive -> /opt/git/bin/post-receive
|   +- post-receive-00logamqp -> /opt/git/bin/post-receive-00logamqp
|   +- post-receive-01email -> /opt/git/bin/post-receive-01email
+- info
|   +- exclude
+- objects
|   +- info
|   +- pack
+- refs
    +- heads
```

```
+-- tags

9 directories, 7 files
```

As can be seen in the output of `tree` at the end, the steps above only create post-receive hooks for logging to AMQP and sending email when a `git push` is done. To add a Jenkins build hook, do the following command as well:

```
git@jira:/opt/git$ ln -s /opt/git/bin/post-receive-02jenkins dims-ocd.git/hooks/post-
↪receive-02jenkins
git@jira:/opt/git$ tree dims-ocd.git/hooks/
dims-ocd.git/hooks/
+- post-receive -> /opt/git/bin/post-receive
+- post-receive-00logamqp -> /opt/git/bin/post-receive-00logamqp
+- post-receive-01email -> /opt/git/bin/post-receive-01email
+- post-receive-02jenkins -> /opt/git/bin/post-receive-02jenkins

0 directories, 4 files
```

Setting up a local Git repository before pushing to remote

After setting up the remote repository, you should create the initial local repository. The basic steps are as follows:

1. Create the new local repo directory.
2. Populate the directory with the files you want in the repo.
3. Add them to the repo.
4. Commit the files with a comment
5. Create an initial version tag.
6. Set `remote.origin.url` to point to the remote repo.
7. Push the new repo to the remote repo.
8. Push the tags to the remote repo.

Here is an edited transcript of performing the above tasks.

```
[dittrich@localhost ~]$ cd $GIT
[dittrich@localhost git]$ mkdir dims-ocd
[dittrich@localhost git]$ git init
Initialized empty Git repository in /Users/dittrich/git/.git/
[ ... prepare files ... ]
[dittrich@localhost dims-ocd (master)]$ ls
MIL-STD-498-templates.pdf  UW-logo.png  conf.py
↪ newsystem.rst
Makefile  _build  currentsystem.rst
↪ notes.rst
OCD-DID.pdf  _static  impacts.rst
↪ operationalscenarios.rst
OCD.html  _templates  index.rst
↪ referenceddocs.rst
OCD.rst  analysis.rst  justifications.rst
↪ scope.rst
UW-logo-32x32.ico  appendices.rst  license.txt
[dittrich@localhost dims-ocd (master)]$ rm OCD.rst
```

```
[dittrich@localhost dims-ocd (master)]$ ls
MIL-STD-498-templates.pdf  _build          currentsystem.rst
↪ notes.rst
Makefile                  _static         impacts.rst
↪ operationalscenarios.rst
OCD-DID.pdf              _templates      index.rst
↪ referenceddocs.rst
OCD.html                 analysis.rst     justifications.rst
↪ scope.rst
UW-logo-32x32.ico        appendices.rst  license.txt
UW-logo.png              conf.py         newssystem.rst
[dittrich@localhost dims-ocd (master)]$ git add .
[dittrich@localhost dims-ocd (master)]$ git commit -m "Initial load of MIL-STD-498_
↪ template"
[master (root-commit) 39816fa] Initial load of MIL-STD-498 template
22 files changed, 1119 insertions(+)
create mode 100644 dims-ocd/MIL-STD-498-templates.pdf
create mode 100644 dims-ocd/Makefile
create mode 100644 dims-ocd/OCD-DID.pdf
create mode 100755 dims-ocd/OCD.html
create mode 100644 dims-ocd/UW-logo-32x32.ico
create mode 100644 dims-ocd/UW-logo.png
create mode 100644 dims-ocd/_build/.gitignore
create mode 100644 dims-ocd/_static/.gitignore
create mode 100644 dims-ocd/_templates/.gitignore
create mode 100644 dims-ocd/analysis.rst
create mode 100644 dims-ocd/appendices.rst
create mode 100644 dims-ocd/conf.py
create mode 100644 dims-ocd/currentsystem.rst
create mode 100644 dims-ocd/impacts.rst
create mode 100644 dims-ocd/index.rst
create mode 100644 dims-ocd/justifications.rst
create mode 100644 dims-ocd/license.txt
create mode 100644 dims-ocd/newssystem.rst
create mode 100644 dims-ocd/notes.rst
create mode 100644 dims-ocd/operationalscenarios.rst
create mode 100644 dims-ocd/referenceddocs.rst
create mode 100644 dims-ocd/scope.rst
[dittrich@localhost dims-ocd (master)]$ git tag -a "2.0.0" -m "Initial template_
↪ release"
[dittrich@localhost dims-ocd (master)]$ git remote add origin git@git.prisem.
↪ washington.edu:/opt/git/dims-ocd.git
[dittrich@localhost dims-ocd (master)]$ git push -u origin master
Counting objects: 24, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (22/22), done.
Writing objects: 100% (24/24), 251.34 KiB | 0 bytes/s, done.
Total 24 (delta 1), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 15 20:46:33 PST 2015
To git@git.prisem.washington.edu:/opt/git/dims-ocd.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin by rebasing.
[dittrich@localhost dims-ocd (master)]$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 173 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 15 20:46:45 PST 2015
To git@git.prisem.washington.edu:/opt/git/dims-ocd.git
```

```
* [new tag]          2.0.0 -> 2.0.0
```

Deleting Sensitive Data From Repos

Before publishing once private repositories on an internal Git repo server to a public server like GitHub requires making sure that **all** sensitive data is permanently removed from the local repository's history **before** first pushing it to GitHub.

Danger: Read what GitHub has to say in the Danger block at the top of their [Remove sensitive data](#) page. In short, any time extremely sensitive data (like a password or private key) is published to GitHub, it should **be considered compromised**, reported to the project lead, and changed as soon as possible.

Follow the instructions in GitHub's [Remove sensitive data](#) page to use either `git filter-branch` or the [BFG Repo-Cleaner](#) to remove files from a clone of the repo and then push the clean version to GitHub.

Cherry-picking a commit from one branch to another

There are times when you are working on one branch (e.g., `feature/coreos`) and find that there is a bug due to a missing file. This file should be on the `develop` branch from which this feature branch was forked, so the solution is to fix the bug on the `develop` branch and also get the fix on the feature branch.

As long as that change (e.g., an added file that does not exist on the branch) has no chance of a conflict, a simple cherry-pick of the commit will get things synchronized. Here is an example of the steps:

Let's say the bug was discovered by noticing this error message shows up when rendering a Sphinx document using `sphinx-autobuild`:

```
+----- source/index.rst changed -----+
/Users/dittrich/git/dims-ci-utils/docs/source/lifecycle.rst:306: WARNING: External
↳Graphviz file u'/Users/dittrich/git/dims-ci-utils/Makefile.dot' not found or
↳reading it failed
+-----+
```

The file `Makefile.dot` is not found. (The reason is that the `lifecycle.rst` file was moved from a different place, but the file it included was not.) We first stash our work (if necessary) and check out the `develop` branch. Next, locate the missing file:

```
[dittrich@localhost docs (feature/coreos)]$ git checkout develop
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.
[dittrich@localhost docs (develop)]$ find ../../ -name 'Makefile.dot'
../../packer/Makefile.dot
```

We now copy the file to where we believe it should reside, and to trigger a new `sphinx-autobuild`, we touch the file that includes it:

```
[dittrich@localhost docs (develop)]$ cp ../../packer/Makefile.dot ..
[dittrich@localhost docs (develop)]$ touch source/lifecycle.rst
```

Switching to the `sphinx-autobuild` status window, we see the error message has gone away.

```
+----- source/lifecycle.rst changed -----
+-----

[I 150331 16:40:04 handlers:74] Reload 1 waiters: None
[I 150331 16:40:04 web:1825] 200 GET /lifecycle.html (127.0.0.1) 0.87ms
[I 150331 16:40:04 web:1825] 200 GET /_static/css/theme.css (127.0.0.1) 1.87ms
[I 150331 16:40:04 web:1825] 304 GET /livereload.js (127.0.0.1) 0.50ms
[I 150331 16:40:04 web:1825] 200 GET /_static/doctools.js (127.0.0.1) 0.43ms
[I 150331 16:40:04 web:1825] 200 GET /_static/jquery.js (127.0.0.1) 0.67ms
[I 150331 16:40:04 web:1825] 200 GET /_static/underscore.js (127.0.0.1) 0.48ms
[I 150331 16:40:04 web:1825] 200 GET /_static/js/theme.js (127.0.0.1) 0.40ms
[I 150331 16:40:04 web:1825] 200 GET /_images/virtual_machine_lifecycle_v2.jpeg (127.
↪0.0.1) 4.61ms
[I 150331 16:40:04 web:1825] 200 GET /_images/whiteboard-lifecycle.png (127.0.0.1) 2.
↪02ms
[I 150331 16:40:04 web:1825] 200 GET /_images/packer_diagram.png (127.0.0.1) 1.65ms
[I 150331 16:40:04 web:1825] 200 GET /_images/screenshot-lifecycle.png (127.0.0.1) 1.
↪37ms
[I 150331 16:40:04 web:1825] 200 GET /_images/vm_org_chart.png (127.0.0.1) 0.70ms
[I 150331 16:40:04 web:1825] 200 GET /_images/graphviz-
↪f8dca63773d709e39ae45240fc6b7ed94229eb74.png (127.0.0.1) 0.92ms
[I 150331 16:40:04 web:1825] 200 GET /_static/fonts/fontawesome-webfont.woff?v=4.0.3_
↪(127.0.0.1) 0.55ms
[I 150331 16:40:05 handlers:109] Browser Connected: http://127.0.0.1:41013/lifecycle.
↪html
```

Now we double-check to make sure we have the change we expect, add, and commit the fix:

```
[dittrich@localhost docs (develop)]$ git stat
?? Makefile.dot
[dittrich@localhost docs (develop)]$ git add ../Makefile.dot
[dittrich@localhost docs (develop)]$ git commit -m "Add Makefile.dot from packer repo_
↪for lifecycle.rst"
[develop d5a948e] Add Makefile.dot from packer repo for lifecycle.rst
1 file changed, 83 insertions(+)
create mode 100644 Makefile.dot
```

Make note of the commit that includes just the new file: commit d5a948e in this case. Now you could bump the version if necessary before pushing.

```
[dittrich@localhost docs (develop)]$ (cd ../; bumpversion patch)
[dittrich@localhost docs (develop)]$ git hf push
Fetching origin
Already up-to-date.
Counting objects: 10, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 783 bytes | 0 bytes/s, done.
Total 10 (delta 8), reused 0 (delta 0)
remote: Running post-receive hook: Tue Mar 31 17:02:43 PDT 2015
remote:  % Total      % Received % Xferd  Average Speed   Time    Time       Time  _
↪Current
remote:
remote:                Dload  Upload  Total  Spent    Left  Speed
remote: 100    217  100    217    0     0   2356      0 --:--:-- --:--:-- --:--:--   2679
remote: Scheduled polling of dims-ci-utils-deploy-develop
remote: Scheduled polling of dims-ci-utils-deploy-master
remote: Scheduled polling of dims-seed-jobs
remote: No git consumers for URI git@git.prisem.washington.edu:/opt/git/dims-ci-utils.
↪git
```

```

remote: [+++] post-receive-06jenkinsalldocs started
remote: [+++] REPONAME=dims-ci-utils
remote: [+++] BRANCH=develop
remote: [+++] newrev=a95c9e1356ff7c6aaed5bcdbe7b533ffc74b6cc1
remote: [+++] oldrev=d5a948ebef61da98b7849416ee340e0a4ba45a3a
remote: [+++] Branch was updated.
remote: [+++] This repo has a documentation directory.
remote:  % Total      % Received % Xferd  Average Speed   Time    Time     Time  ┐
↪Current
remote:
remote:      Dload  Upload  Total  Spent    Left  Speed
remote: 100    79    0    0 100    79      0  1359  --:--:-- --:--:-- --:--:-- 1612
remote:  % Total      % Received % Xferd  Average Speed   Time    Time     Time  ┐
↪Current
remote:
remote:      Dload  Upload  Total  Spent    Left  Speed
remote: 100    78    0    0 100    78      0   260  --:--:-- --:--:-- --:--:-- 268
remote: [+++] post-receive-06jenkinsalldocs finished
To git@git.prisem.washington.edu:/opt/git/dims-ci-utils.git
    d5a948e..a95c9e1  develop -> develop

Summary of actions:
- The remote branch 'origin/develop' was updated with your changes

```

Now you can go back to the feature branch you were working on, and cherry-pick the commit with the missing file.

```

[dittrich@localhost docs (develop)]$ git checkout feature/coreos
Switched to branch 'feature/coreos'
Your branch is ahead of 'origin/feature/coreos' by 1 commit.
  (use "git push" to publish your local commits)
[dittrich@localhost docs (feature/coreos)]$ git cherry-pick d5a948e
[feature/coreos 14dbf59] Add Makefile.dot from packer repo for lifecycle.rst
Date: Tue Mar 31 16:38:03 2015 -0700
1 file changed, 83 insertions(+)
create mode 100644 Makefile.dot
[dittrich@localhost docs (feature/coreos)]$ git log
commit 14dbf59dff5d6fce51c899b32fef87276dbddef7
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Tue Mar 31 16:38:03 2015 -0700

    Add Makefile.dot from packer repo for lifecycle.rst
...

```

Note: Note that this results in a new commit hash on this branch (in this case, 14dbf59dff5d6fce51c899b32fef87276dbddef7).

Synchronizing with an *upstream* repository

Note: The DIMS project is using forks of several source repositories, some for the sake of local customization, and some for adding features necessary for DIMS purposes. The [MozDef](#) project is one of these (see the [dimsad:dimsarchitecturedesign](#) document, Section [dimsad:conceptofexecution](#)).

To track another project's Git repository, syncing it with a fork that you use locally, it is necessary to do the following:

- [Configuring a remote for a fork](#)

- Syncing a fork

1. Make sure that you have defined *upstream* properly, e.g.,

```
[dimsenv] ~/dims/git/MozDef (master) $ git remote -v
origin      git@git.prisem.washington.edu:/opt/git/MozDef.git (fetch)
origin      git@git.prisem.washington.edu:/opt/git/MozDef.git (push)
upstream    git@github.com:jeffbryner/MozDef.git (fetch)
upstream    git@github.com:jeffbryner/MozDef.git (push)
```

2. Fetch the contents of the upstream remote repository:

```
[dimsenv] ~/dims/git/MozDef (master) $ git fetch upstream
remote: Counting objects: 6, done.
remote: Total 6 (delta 2), reused 2 (delta 2), pack-reused 4
Unpacking objects: 100% (6/6), done.
From github.com:jeffbryner/MozDef
   700c1be..4575c0f  master    -> upstream/master
* [new tag]         v1.12     -> v1.12
```

3. Checkout the branch to sync (e.g., master) and then merge any changes:

```
[dimsenv] ~/dims/git/MozDef (master) $ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.
[dimsenv] ~/dims/git/MozDef (master) $ git merge upstream/master
Merge made by the 'recursive' strategy.
  alerts/unauth_ssh_pyes.conf | 4 ++++
  alerts/unauth_ssh_pyes.py   | 78_
↪ ++++++
2 files changed, 82 insertions(+)
 create mode 100644 alerts/unauth_ssh_pyes.conf
 create mode 100644 alerts/unauth_ssh_pyes.py
[dimsenv] ~/dims/git/MozDef (master) $ git push origin master
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 2.11 KiB | 0 bytes/s, done.
Total 8 (delta 3), reused 0 (delta 0)
remote: Running post-receive hook: Thu Sep 17 20:52:14 PDT 2015
To git@git.prisem.washington.edu:/opt/git/MozDef.git
   180484a..766da56  master -> master
```

4. Now push the updated repository to the “local” *remote repository* (i.e, git.prisem.washington.edu for the DIMS project):

```
[dimsenv] ~/dims/git/MozDef (master) $ git push origin master
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 2.11 KiB | 0 bytes/s, done.
Total 8 (delta 3), reused 0 (delta 0)
remote: Running post-receive hook: Thu Sep 17 20:52:14 PDT 2015
To git@git.prisem.washington.edu:/opt/git/MozDef.git
   180484a..766da56  master -> master
```

5. If the *remote repository* is itself the fork (e.g., if you fork a repository on GitHub, then want to maintain a “local” *remote repository* on-site for your project, you may wish to use a label other than *upstream* to connote the fork differently):


```
[dimesenv] ~/git/ansible (release1.8.4*) $ git remote -v
davedittrich      git@github.com:davedittrich/ansible.git (fetch)
davedittrich      git@github.com:davedittrich/ansible.git (push)
origin            https://github.com/ansible/ansible.git (fetch)
origin            https://github.com/ansible/ansible.git (push)
```

Starting a “release”

By convention, DIMS repositories have at least one file, named `VERSION`, that contains the release version number. You can see the current release by looking at the contents of this file.

```
[dittrich@localhost ansible-playbooks (dev)]$ cat VERSION
1.1.4
```

Note: There may be other files, such as the Sphinx documentation configuration file, `docs/source/conf.py` usually, or other source files for Python or Java builds. Each of the files that has a version/release number in it **must** use the same string and be included in the `.bumpversion.cfg` file in order for `bumpversion` to properly manage release numbers.

Now that you know what the current version number is, you can initiate a release branch with `hub-flow`, knowing that the new numbr will be. In this case, we will create a release branch `1.2.0` to increment the minor version number component.

```
[dittrich@localhost ansible-playbooks (dev)]$ git hf release start 1.2.0
Fetching origin
Switched to a new branch 'release/1.2.0'
Total 0 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:33:54 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
 * [new branch]      release/1.2.0 -> release/1.2.0

Summary of actions:
- A new branch 'release/1.2.0' was created, based on 'dev'
- The branch 'release/1.2.0' has been pushed up to 'origin/release/1.2.0'
- You are now on branch 'release/1.2.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

    git hf release finish '1.2.0'
```

You should now be on the new release branch:

```
[dittrich@localhost ansible-playbooks (release/1.2.0)]$
```

After making any textual changes, bump the version number to match the new release number:

```
[dittrich@localhost ansible-playbooks (release/1.2.0)]$ bumpversion minor
```

Now the release can be finished. You will be placed in an editor to create comments for actions like merges and tags.

```
[dittrich@localhost ansible-playbooks (release/1.2.0)]$ bumpversion minor
[dittrich@localhost ansible-playbooks (release/1.2.0)]$ cat VERSION
1.2.0
[dittrich@localhost ansible-playbooks (release/1.2.0)]$ git hf release finish '1.2.0'
Fetching origin
Fetching origin
Counting objects: 9, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 690 bytes | 0 bytes/s, done.
Total 9 (delta 7), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:37:24 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
   3ac28a2..5ca145b  release/1.2.0 -> release/1.2.0
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
Removing roles/tomcat/tasks/main.yml
Removing roles/tomcat/handlers/main.yml
Removing roles/tomcat/defaults/main.yml
Removing roles/postgres/templates/pg_hba.conf.j2
Removing roles/postgres/files/schema.psqli
Removing roles/ozone/files/postgresql-9.3-1102.jdbc41.jar
Auto-merging roles/logstash/files/demo.logstash.deleteESDB
Auto-merging roles/logstash/files/demo.logstash.addwebsense
Auto-merging roles/logstash/files/demo.logstash.addufw
Auto-merging roles/logstash/files/demo.logstash.addrpcflow
Auto-merging roles/logstash/files/demo.logstash.addcymru

[ ... ]

~
".git/MERGE_MSG" 7L, 280C written
Merge made by the 'recursive' strategy.
 .bumpversion.cfg | 11 +
 Makefile | 61 +-
 VERSION | 1 +
 configure-all.yml | 5 +-
 dims-all-desktop.yml | 56 +
 dims-all-server.yml | 125 ++
 dims-cifv1-server.yml | 50 +

[...]

Release 1.2.0.
#
# Write a message for tag:
# 1.2.0
# Lines starting with '#' will be ignored.

[...]

~
".git/TAG_EDITMSG" 5L, 97C written
Switched to branch 'dev'
Your branch is up-to-date with 'origin/dev'.

Merge tag '1.2.0' into dev for
```

```

Merge tag '1.2.0' into dev for
Merge tag '1.2.0' into dev for Release 1.2.0.

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.

[...]

".git/MERGE_MSG" 7L, 273C written
Merge made by the 'recursive' strategy.
 .bumpversion.cfg      | 2 +-
 VERSION               | 2 +-
 docs/source/conf.py   | 4 +--
 group_vars/all        | 2 +-
 4 files changed, 5 insertions(+), 5 deletions(-)
Deleted branch release/1.2.0 (was 5ca145b).
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 447 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:38:17 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
   3ac28a2..aec921c  dev -> dev
Total 0 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:38:19 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
   2afb58f..2482d07  master -> master
Counting objects: 1, done.
Writing objects: 100% (1/1), 166 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:38:25 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
 * [new tag]           1.2.0 -> 1.2.0
remote: Running post-receive hook: Thu Jan 22 18:38:28 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
 - [deleted]           release/1.2.0

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged '1.2.0'
- Tag '1.2.0' has been back-merged into 'dev'
- Branch 'master' has been back-merged into 'dev'
- Release branch 'release/1.2.0' has been deleted
- 'dev', 'master' and tags have been pushed to 'origin'
- Release branch 'release/1.2.0' in 'origin' has been deleted.

```

Lastly, bump the patch version number in the dev branch to make sure that when something reports the version in developmental code builds, it doesn't look like you are using code from the *last tagged* master branch. That completely defeats the purpose of using version numbers for dependency checks or debugging.

```

[dittrich@localhost ansible-playbooks (dev)]$ bumpversion patch
[dittrich@localhost ansible-playbooks (dev)]$ git push
Counting objects: 9, done.

```

```
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 683 bytes | 0 bytes/s, done.
Total 9 (delta 7), reused 0 (delta 0)
remote: Running post-receive hook: Thu Jan 22 18:51:00 PST 2015
To git@git.prisem.washington.edu:/opt/git/ansible-playbooks.git
aec921c..d4fe053 dev -> dev
```

Figure *New 1.2.0 release on master, dev now on 1.2.1.* shows what the branches look like with GitX.app on a Mac:

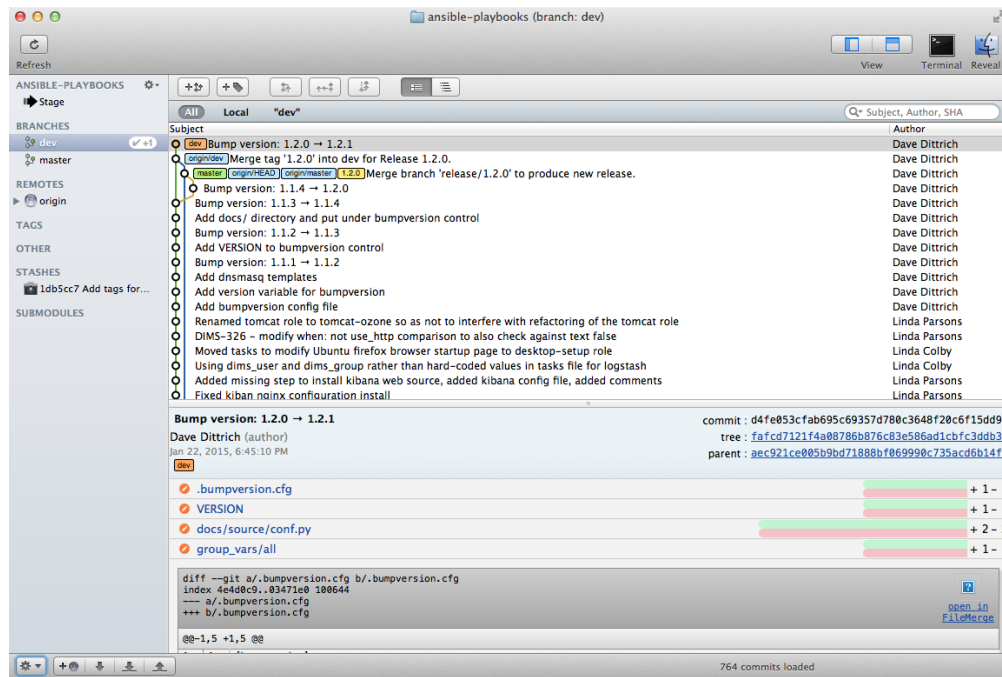


Fig. 4.2: New 1.2.0 release on master, dev now on 1.2.1.

Branch Renaming

Several of the git repos comprising the DIMS source code management system are using the name `dev` for the main development branch. The (somewhat) accepted name for the development branch is `develop`, as detailed in e.g. <http://nvie.com/posts/a-successful-git-branching-model/>.

We would therefore like to rename any `dev` branch to `develop` throughout our git repo set. This will of course impact team members who use the central repos to share work. Research online suggests that branch renaming can be done. The best source found was <https://gist.github.com/lttlrck/9628955>, who suggested a three-part operation

```
# Rename branch locally
git branch -m old_branch new_branch
# Delete the old branch
git push origin :old_branch
# Push the new branch, set local branch to track the new remote
git push --set-upstream origin new_branch
```

To test this recipe out without impacting any existing repos and therefore avoiding any possible loss of real work, we constructed a test situation with a central repo and two fake ‘users’ who both push and pull from that repo. A branch rename is then done, following the recipe above. The impact on each of the two users is noted.

First, we create a bare repo. This will mimic our authoritative repos on `git.prisem.washington.edu`. We'll call this repo `dims-328.git`, named after the DIMS Jira ticket created to study the branch rename issue:

```
$ cd
$ mkdir depot
$ cd depot
$ git init --bare dims-328.git
```

Next, we clone this repo a first time, which simulates the first 'user' (replace `/home/stuart/` with your local path):

```
$ cd
$ mkdir scratch
$ cd scratch
$ git clone file:///home/stuart/depot/dims-328.git
```

Next, we add some content in master branch

```
$ cd dims-328
$ echo content > foo
$ git add foo
$ git commit -m "msg"
$ git push origin master
```

We now clone the 'depot' repo a second time, to simulate the second user. Both users are then developing using the authoritative repo as the avenue to share work. Notice how the second user clones into the specified directory `dims-328-2`, so as not to tread on the first user's work:

```
$ cd ~/scratch
$ git clone file:///home/stuart/depot/dims-328.git dims-328-2
```

user1 (first clone) then creates a dev branch and adds some content to it:

```
$ cd ~/scratch/dims-328
$ git branch dev
$ git checkout dev
$ echo content > devbranch
$ git add devbranch
$ git commit -m "added content to dev branch"
$ git push origin dev
```

This will create a dev branch in the origin repo, i.e the depot.

Next, as the second user, pull the changes, checkout dev and edit:

```
$ cd ~/scratch/dims-328-2
$ git pull
$ git checkout dev
$ echo foo >> devbranch
```

At this point we have two 'users' with local repos, both of which share a common upstream repo. Both users have got the dev branch checked out, and may have local changes on that branch.

Now, we wish to rename the branch dev to develop throughout, i.e. at the depot and in users' repos.

Using instructions from <https://gist.github.com/lttlrck/9628955>, and noting the impacts to each user, we first act as *user1*, who will be deemed 'in charge' of the renaming process:

```
$ cd ~/scratch/dims-328
$ git branch -m dev develop
```

```
$ git push origin :dev
To file:///home/stuart/depot/dims-328.git
- [deleted]          dev
$ git push --set-upstream origin develop
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 259 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To file:///home/stuart/depot/dims-328.git
 * [new branch]      develop -> develop
Branch develop set up to track remote branch develop from origin.
```

Warning: (This reads like a ..warning block. Is that how it was meant?)

The git push output message implies a deletion of the dev branch in the depot. If *user2* were to interact with origin/dev now, what would happen??

Here are the contents of *user1*'s .git/config after the 3-operation rename:

```
[stuart@rejewski dims-328 (develop)]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = file:///home/stuart/depot/dims-328.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[branch "develop"]
    remote = origin
    merge = refs/heads/develop
```

Note how there are references to develop but none to dev. As far as *user1* is concerned, the branch rename appears to have worked and is complete.

Now, what does *user2* see? With dev branch checked out, *and* with a local mod, we do a pull:

```
$ cd ~scratch/dims-328-2
$ git pull
From file:///home/stuart/depot/dims-328
 * [new branch]      develop -> origin/develop
Your configuration specifies to merge with the ref 'dev'
from the remote, but no such ref was fetched.
```

This is some form of error message. *user2*'s .git/config at this point is this:

```
[stuart@rejewski dims-328-2 (dev)]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
```

```
[remote "origin"]
    url = file:///home/stuart/depot/dims-328.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[branch "dev"]
    remote = origin
    merge = refs/heads/dev
```

Perhaps just the branch rename will work for *user2*? As *user2*, we do the first part of the *rename recipe*:

```
$ git branch -m dev develop
```

No errors from this, but *user2*'s `.git/config` still refers to a dev branch:

```
[stuart@rejewski dims-328-2 (dev)]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = file:///home/stuart/depot/dims-328.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[branch "develop"]
    remote = origin
    merge = refs/heads/dev
```

Next, as *user2*, we issued the third part of the *rename recipe* (but skipped the second part):

```
$ git push --set-upstream origin develop
Branch develop set up to track remote branch develop from origin.
Everything up-to-date.
```

Note that this is a push, but since *user2* had no committed changes locally, no content was actually pushed.

Now *user2*'s `.git/config` looks better, the token dev has changed to develop:

```
[stuart@rejewski dims-328-2 (dev)]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = file:///home/stuart/depot/dims-328.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[branch "develop"]
    remote = origin
    merge = refs/heads/develop
```

Next, as *user2*, commit the local change, and push to depot:

```
$ git add devbranch
$ git commit -m "msg"
$ git push
```

So it appears that *user2* can issue just the branch rename and upstream operation, and skip the second component of the 3-part recipe (`git push origin :old_branch`), likely since this is an operation on the remote (depot) itself and was already done by *user1*.

Finally, we switch back to *user1* and pull changes made by *user2*:

```
$ cd ~scratch/dims-328
$ git pull
```

Warning: This has addressed *only* git changes. The wider implications of a git branch rename on systems such as Jenkins has yet to be addressed. Since systems like Jenkins generally just clone or pull from depots, it is expected that only git URLs need to change from including `dev` to `develop`.

Deleting accidentally created tags

When trying to finish a release, you may accidentally create a tag named `finish`. It may even get propagated automatically to `origin`, in which case it could propagate to others' repos:

```
mr update: /Users/dittrich/dims/git/dims-keys
Fetching origin
From git.prisem.washington.edu:/opt/git/dims-keys
* [new tag]          finish      -> finish
```

You can delete them locally and remotely with the following commands:

```
[dittrich@localhost dims-keys (develop)]$ git tag -d finish
Deleted tag 'finish' (was 516d9d2)
[dittrich@localhost dims-keys (develop)]$ git push origin :refs/tags/finish
remote: Running post-receive hook: Thu Aug 6 16:07:17 PDT 2015
To git@git.prisem.washington.edu:/opt/git/dims-keys.git
- [deleted]          finish
```

Recovering deleted files

Files that have been deleted in the past, and the deletions committed, can be recovered by searching the Git history of deletions to identify the commit that included the deletion. The file can then be checked out using the predecessor to that commit. See [Find and restore a deleted file in a Git repository](#)

Fixing comments in unpublished commits

Note: This section was derived from <http://makandracards.com/makandra/868-change-commit-messages-of-past-git-commits>

Warning: Only do this if you have **not already pushed** the changes!! As noted in the `git-commit` man page for the `--amend` option:

```
You should understand the implications of rewriting history if you
amend a commit that has already been published. (See the "RECOVERING
FROM UPSTREAM REBASE" section in git-rebase(1).)
```

There may be times when you accidentally make multiple commits, one at a time, using the same comment (but the changes are not related to the comment).

Here is an example of three commits all made with `git commit -am` using the same message:

```
(dimsenv)[dittrich@localhost docs (develop)]$ git log
commit 08b888b9dd33f53f0e26d8ff8aab7309765ad0eb
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Thu Apr 30 18:35:08 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

commit 7f3d0d8134c000a787aad83f2690808008ed1d96
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Thu Apr 30 18:34:40 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

commit f6f5d868c8ddd12018ca662a54d1f58c150e6364
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Thu Apr 30 18:33:59 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

commit 96575c967f606e2161033de92dd2dc580ad60a8b
Merge: 1253ea2 dae5aca
Author: Linda Parsons <lparsonstech@gmail.com>
Date: Thu Apr 30 14:00:49 2015 -0400

    Merge remote-tracking branch 'origin/develop' into develop

commit 1253ea20bc553759c43d3a999b81be009851d195
Author: Linda Parsons <lparsonstech@gmail.com>
Date: Thu Apr 30 14:00:19 2015 -0400

    Added information for deploying to infrastructure
```

Note: Make note that the commit immediately prior to the three erroneously commented commits is 96575c96. We will use that commit number in a moment...

Looking at the patch information shows these are clearly not all correctly commented:

```
(dimsenv)[dittrich@localhost docs (develop)]$ git log --patch
commit 08b888b9dd33f53f0e26d8ff8aab7309765ad0eb
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Thu Apr 30 18:35:08 2015 -0700
```

```

    Fix intersphinx links to use DOCSURL env variable

diff --git a/docs/makedocset b/docs/makedocset
index dafbedb..9adb954 100644
--- a/docs/makedocset
+++ b/docs/makedocset
@@ -7,7 +7,14 @@
    # This is useful for building a set of documents that employ
    # intersphinx linking, obtaining the links from the co-local
    # repositories instead of specified remote locations.
+#
+# To build the docs for a specific server (e.g., when building
+# using a local docker container running Nginx), set the
+# environment variable DOCSURL to point to the server:
+#
+# $ export DOCSURL=http://192.168.99.100:49153

+DOCSURL=${DOCSURL:-http://ul2-dev-svr-1.prisem.washington.edu:8080/docs/devel}

    # Activate dimsenv virtual environment for Sphinx
    . $HOME/dims/envs/dimsenv/bin/activate

commit 7f3d0d8134c000a787aad83f2690808008ed1d96
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:34:40 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

diff --git a/docs/source/conf.py b/docs/source/conf.py
index 9fdc100..b3cd483 100644
--- a/docs/source/conf.py
+++ b/docs/source/conf.py
@@ -351,13 +351,16 @@
@@ epub_exclude_files = ['search.html']
    # If false, no index is generated.
    #epub_use_index = True

+os.environ['GITBRANCH'] = "develop"
+
+if os.environ.get('DOCSURL') is None:
+    #os.environ['DOCSURL'] = "file://{}/".format(os.environ.get('GIT'))
+    os.environ['DOCSURL'] = "http://ul2-dev-svr-1.prisem.washington.edu:8080/docs/{}/
+→html/".format(
+    os.environ['GITBRANCH'])

    intersphinx_cache_limit = -1    # days to keep the cached inventories (0 == forever)
    intersphinx_mapping = {
-        'dimsocd': ("%s/dims/docs/dims-ocd" % os.environ['HOME'],
-                    ('http://ul2-dev-svr-1.prisem.washington.edu:8080/docs/develop/
+→html/dims-ocd/objects.inv', None)),
-        'dimsad': ("%s/dims/docs/dims-ad" % os.environ['HOME'],
-                    ('http://ul2-dev-svr-1.prisem.washington.edu:8080/docs/develop/
+→html/dims-ad/objects.inv', None)),
-        'dimssr': ("%s/dims/docs/dims-sr" % os.environ['HOME'],
-                    ('http://ul2-dev-svr-1.prisem.washington.edu:8080/docs/develop/
+→html/dims-sr/objects.inv', None))
+        'dimsocd': ("{} /dims-ocd".format(os.environ['DOCSURL']), None),
+        'dimsad': ("{} /dims-ad".format(os.environ['DOCSURL']), None),
+        'dimssr': ("{} /dims-sr".format(os.environ['DOCSURL']), None)

```

```

}

commit f6f5d868c8ddd12018ca662a54d1f58c150e6364
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:33:59 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

diff --git a/docs/makedocs b/docs/makedocs
deleted file mode 100644
index dafbedb..0000000
--- a/docs/makedocs
+++ /dev/null
@@ -1,66 +0,0 @@
-#!/bin/bash -x
-#
-# This script builds multiple Sphinx documents in repos
-# residing (in their current checkout branch/state) in
-# the directory specified by the $GIT environment variable.
-#
-# This is useful for building a set of documents that employ
-# intersphinx linking, obtaining the links from the co-local
-# repositories instead of specified remote locations.
...

```

The last commit is easy to fix. Just use `git commit --amend` and edit the message:

```

(dimsenv)[dittrich@localhost docs (develop)]$ git commit --amend

Add DOCSURL selection of where docs reside for intersphinx links

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Apr 30 18:35:08 2015 -0700
#
# On branch develop
# Your branch is ahead of 'origin/develop' by 3 commits.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#       modified:   makedocset

```

Now we can see the message has been changed, but so has the commit hash!

```

(dimsenv)[dittrich@localhost docs (develop)]$ git log --patch
commit 654cb34378cb0a4140725a37e3724b6dcee7aebd
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:35:08 2015 -0700

    Add DOCSURL selection of where docs reside for intersphinx links

diff --git a/docs/makedocset b/docs/makedocset
index dafbedb..9adb954 100644
--- a/docs/makedocset
+++ b/docs/makedocset
@@ -7,7 +7,14 @@
 # This is useful for building a set of documents that employ

```

```
# intersphinx linking, obtaining the links from the co-local
# repositories instead of specified remote locations.
+#
+# To build the docs for a specific server (e.g., when building
+# using a local docker container running Nginx), set the
+# environment variable DOCSURL to point to the server:
+#
+# $ export DOCSURL=http://192.168.99.100:49153

+DOCSURL=${DOCSURL:-http://u12-dev-svr-1.prisem.washington.edu:8080/docs/devel}

# Activate dimsenv virtual environment for Sphinx
. $HOME/dims/envs/dimsenv/bin/activate

commit 7f3d0d8134c000a787aad83f2690808008ed1d96
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date: Thu Apr 30 18:34:40 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

diff --git a/docs/source/conf.py b/docs/source/conf.py
...
```

The second commit has the correct comment, but commit `f6f5d868c` was simply renaming a file. It got caught up as a commit when the `-a` option was given when committing the changed file, not realizing the renamed file had already been added to the cache.

To change the message for *only* commit `f6f5d86`, start an interactive rebase at the commit immediately prior to that commit (in this case, commit `96575c9`). Change pick to edit for that commit.

```
(dimsenv)[dittrich@localhost docs (develop)]$ git rebase -i 96575c9

edit f6f5d86 Fix intersphinx links to use DOCSURL env variable
pick 7f3d0d8 Fix intersphinx links to use DOCSURL env variable
pick 654cb34 Add DOCSURL selection of where docs reside for intersphinx links

# Rebase 96575c9..654cb34 onto 96575c9 (      3 TODO item(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

As soon as you exit the editor, Git will begin the rebase and tell you what to do next:

```
Stopped at f6f5d868c8ddd12018ca662a54d1f58c150e6364... Fix intersphinx links to use_
↪DOCSURL env variable
```

You can amend the commit now, with

```
git commit --amend
```

Once you are satisfied with your changes, run

```
git rebase --continue
```

Now use `git commit --amend` to edit the comment:

```
(dimsenv)[dittrich@localhost docs (develop|REBASE-i 1/3)]$ git commit --amend

Rename makedocs -> makedocset

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Apr 30 18:33:59 2015 -0700
#
# rebase in progress; onto 96575c9
# You are currently editing a commit while rebasing branch 'develop' on '96575c9'.
#
# Changes to be committed:
#       renamed:    makedocs -> makedocset
#
```

Finish off by continuing the rebase for the remaining commits.

```
(dimsenv)[dittrich@localhost docs (develop|REBASE-i 1/3)]$ git rebase --continue
Successfully rebased and updated refs/heads/develop.
```

Now `git log` shows the correct comments, as well as new commit hashes:

```
(dimsenv)[dittrich@localhost docs (develop)]$ git log
commit 89af6d9fda07276d3cb06dfd2977f1392fb03b25
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:35:08 2015 -0700

    Add DOCSURL selection of where docs reside for intersphinx links

commit c2c55ff3dcbf10739c5d86ce8a6192e930ccd265
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:34:40 2015 -0700

    Fix intersphinx links to use DOCSURL env variable

commit 2155936ad7e3ae71ef5775b2036a4b6c21a9a86d
Author: Dave Dittrich <dave.dittrich@gmail.com>
Date:   Thu Apr 30 18:33:59 2015 -0700

    Rename makedocs -> makedocset

commit 96575c967f606e2161033de92dd2dc580ad60a8b
Merge: 1253ea2 dae5aca
Author: Linda Parsons <lparsonstech@gmail.com>
Date:   Thu Apr 30 14:00:49 2015 -0400

    Merge remote-tracking branch 'origin/develop' into develop
```

Creating a new documentation-only repo

Note: TBD

The following is included here to document how to set up a new documentation-only repo. The lines that are highlighted are those that include user input. The long-term goal is to script creating these repos so as to not require everyone know exactly how to answer each of these questions. This is blocked waiting on getting a consistent Python virtual environment that works on both dev systems and Jenkins before globally functional scripts and Sphinx configurations will work properly.

```
1 [dittrich@localhost git]$ mkdir dims-asbuilt
2 [dittrich@localhost git]$ cd dims-asbuilt/
3 [dittrich@localhost dims-asbuilt]$ git init
4 Initialized empty Git repository in /Users/dittrich/git/dims-asbuilt/.git/
5 [dittrich@localhost dims-asbuilt (master)]$ workon dimsenv
6 (dimsenv)[dittrich@localhost dims-asbuilt (master)]$ sphinx-quickstart
7 Welcome to the Sphinx 1.3.1 quickstart utility.
8
9 Please enter values for the following settings (just press Enter to
10 accept a default value, if one is given in brackets).
11
12 Enter the root path for documentation.
13 > Root path for the documentation [.]:
14
15 You have two options for placing the build directory for Sphinx output.
16 Either, you use a directory "_build" within the root path, or you separate
17 "source" and "build" directories within the root path.
18 > Separate source and build directories (y/n) [n]: y
19
20 Inside the root directory, two more directories will be created; "_templates"
21 for custom HTML templates and "_static" for custom stylesheets and other static
22 files. You can enter another prefix (such as ".") to replace the underscore.
23 > Name prefix for templates and static dir [_]:
24
25 The project name will occur in several places in the built documentation.
26 > Project name: DIMS 'As-Built' System
27 > Author name(s): Dave Dittrich
28
29 Sphinx has the notion of a "version" and a "release" for the
30 software. Each version can have multiple releases. For example, for
31 Python the version is something like 2.5 or 3.0, while the release is
32 something like 2.5.1 or 3.0a1. If you don't need this dual structure,
33 just set both to the same value.
34 > Project version: 0.1.0
35 > Project release [0.1.0]:
36
37 If the documents are to be written in a language other than English,
38 you can select a language here by its language code. Sphinx will then
39 translate text that it generates into that language.
40
41 For a list of supported codes, see
42 http://sphinx-doc.org/config.html#confval-language.
43 > Project language [en]:
44
45 The file name suffix for source files. Commonly, this is either ".txt"
46 or ".rst". Only files with this suffix are considered documents.
```

```

47 > Source file suffix [.rst]:
48
49 One document is special in that it is considered the top node of the
50 "contents tree", that is, it is the root of the hierarchical structure
51 of the documents. Normally, this is "index", but if your "index"
52 document is a custom template, you can also set this to another filename.
53 > Name of your master document (without suffix) [index]:
54
55 Sphinx can also add configuration for epub output:
56 > Do you want to use the epub builder (y/n) [n]: y
57
58 Please indicate if you want to use one of the following Sphinx extensions:
59 > autodoc: automatically insert docstrings from modules (y/n) [n]:
60 > doctest: automatically test code snippets in doctest blocks (y/n) [n]:
61 > intersphinx: link between Sphinx documentation of different projects (y/n) [n]: y
62 > todo: write "todo" entries that can be shown or hidden on build (y/n) [n]: y
63 > coverage: checks for documentation coverage (y/n) [n]:
64 > pngmath: include math, rendered as PNG images (y/n) [n]:
65 > mathjax: include math, rendered in the browser by MathJax (y/n) [n]:
66 > ifconfig: conditional inclusion of content based on config values (y/n) [n]: y
67 > viewcode: include links to the source code of documented Python objects (y/n) [n]:
68
69 A Makefile and a Windows command file can be generated for you so that you
70 only have to run e.g. `make html` instead of invoking sphinx-build
71 directly.
72 > Create Makefile? (y/n) [y]:
73 > Create Windows command file? (y/n) [y]: n
74
75 Creating file ./source/conf.py.
76 Creating file ./source/index.rst.
77 Creating file ./Makefile.
78
79 Finished: An initial directory structure has been created.
80
81 You should now populate your master file ./source/index.rst and create other_
82 ↳documentation
83 source files. Use the Makefile to build the docs, like so:
84     make builder
85 where "builder" is one of the supported builders, e.g. html, latex or linkcheck.
86
87 (dimsenv)[dittrich@localhost dims-asbuilt (master)]$ echo \
88 > "This is a documentation-only repo. Sphinx source is in docs/source." > README.txt
89 (dimsenv)[dittrich@localhost dims-asbuilt (master)]$ tree
90 .
91 +- README.txt
92 +- Makefile
93 +- build
94 +- source
95     +- _static
96     +- _templates
97     +- conf.py
98     +- index.rst
99
100 4 directories, 4 files
101 (dimsenv)[dittrich@localhost dims-asbuilt (master)]$ dims.sphinx-autobuild
Serving on http://127.0.0.1:29583

```

After setting up the directory structure, editing the source/conf.py file to fix the title, etc., and creating initial

scaffolding files sufficient to render a Sphinx document, you are almost ready to commit to Git. First, do make clean to get rid of any rendered files and make sure that only the source files and `README.txt` file are present:

```
[dittrich@localhost dims-asbuilt (master)]$ make clean
rm -rf build/*
[dittrich@localhost dims-asbuilt (master)]$ tree
.
+- Makefile
+- README.txt
+- build
+- source
    +- _static
    +- _templates
    +- cifv1.rst
    +- conf.py
    +- git.rst
    +- index.rst
    +- jenkins.rst

4 directories, 7 files
```

The next step is to add the source to the local git repo, set the upstream origin, tag the repository with the version number specified above, and push it to origin.

```
[dittrich@localhost dims-asbuilt (master)]$ git add .
[dittrich@localhost dims-asbuilt (master)]$ git stat
A Makefile
A README.txt
A source/cifv1.rst
A source/conf.py
A source/git.rst
A source/index.rst
A source/jenkins.rst
[dittrich@localhost dims-asbuilt (master)]$ git commit -m "Initial load"
[master (root-commit) d0fcaa5] Initial load
 7 files changed, 604 insertions(+)
 create mode 100644 Makefile
 create mode 100644 README.txt
 create mode 100644 source/cifv1.rst
 create mode 100644 source/conf.py
 create mode 100644 source/git.rst
 create mode 100644 source/index.rst
 create mode 100644 source/jenkins.rst
[dittrich@localhost dims-asbuilt (master)]$ git remote add origin git@git.prisem.
↪washington.edu:/opt/git/dims-asbuilt.git
[dittrich@localhost dims-asbuilt (master)]$ git tag -a "0.1.0" -m "Initial template_
↪release"
[dittrich@localhost dims-asbuilt (master)]$ git push origin master
Counting objects: 10, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 7.37 KiB | 0 bytes/s, done.
Total 10 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Wed Mar 18 16:15:02 PDT 2015
To git@git.prisem.washington.edu:/opt/git/dims-asbuilt.git
 * [new branch]      master -> master
[dittrich@localhost dims-asbuilt (master)]$ git push origin --tags
Counting objects: 1, done.
```



```
Writing objects: 100% (1/1), 173 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Wed Mar 18 16:26:29 PDT 2015
To git@git.prisem.washington.edu:/opt/git/dims-asbuilt.git
 * [new tag]          0.1.0 -> 0.1.0
```

Following those steps, initialize the repo for hub-flow.

```
[dittrich@localhost dims-asbuilt (master)]$ git hf init
Using default branch names.

Which branch should be used for tracking production releases?
- master
  Branch name for production releases: [master]
  Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Total 0 (delta 0), reused 0 (delta 0)
remote: Running post-receive hook: Wed Mar 18 16:24:14 PDT 2015
To git@git.prisem.washington.edu:/opt/git/dims-asbuilt.git
 * [new branch]       develop -> develop
```

Set up bumpversion:

```
[dittrich@localhost dims-asbuilt (develop)]$ vi .bumpversion.cfg

[bumpversion]
current_version = 0.1.0
commit = True
tag = False

[bumpversion:file:source/conf.py]
```

Use the `--dry-run` option to test whether the configuration was done properly before attempting to actually bump the version number.

```
[dittrich@localhost dims-asbuilt (develop)]$ bumpversion --dry-run --verbose patch
Reading config file .bumpversion.cfg:
[bumpversion]
current_version = 0.1.0
commit = True
tag = False

[bumpversion:file:source/conf.py]

Parsing version '0.1.0' using regexp '(?P<major>\d+)\. (?P<minor>\d+)\. (?P<patch>\d+)'
Parsed the following values: major=0, minor=1, patch=0
Attempting to increment part 'patch'
Values are now: major=0, minor=1, patch=1
Dry run active, won't touch any files.
New version will be '0.1.1'
Asserting files source/conf.py contain the version string:
```

```
Found '0.1.0' in source/conf.py at line 61: version = '0.1.1'
Would change file source/conf.py:
--- a/source/conf.py
+++ b/source/conf.py
@@ -59,9 +59,9 @@
 # built documents.
 #
 # The short X.Y version.
-version = '0.1.0'
+version = '0.1.1'
 # The full version, including alpha/beta/rc tags.
-release = '0.1.0'
+release = '0.1.1'

 # The language for content autogenerated by Sphinx. Refer to documentation
 # for a list of supported languages.
Would write to config file .bumpversion.cfg:
[bumpversion]
current_version = 0.1.1
commit = True
tag = False

[bumpversion:file:source/conf.py]

Would prepare Git commit
Would add changes in file 'source/conf.py' to Git
Would add changes in file '.bumpversion.cfg' to Git
Would commit to Git with message 'Bump version: 0.1.0 → 0.1.1'
Would tag 'v0.1.1' in Git
[dittrich@localhost dims-asbuilt (develop)]$ bumpversion patch
```

Now use hub-flow to push the current state of the local repo.

```
[dittrich@localhost dims-asbuilt (develop)]$ git hf push
Fetching origin
Already up-to-date.
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 375 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Running post-receive hook: Wed Mar 18 16:38:27 PDT 2015
To git@git.prisem.washington.edu:/opt/git/dims-asbuilt.git
    d0fcaa5..db3c7f1  develop -> develop

Summary of actions:
- The remote branch 'origin/develop' was updated with your changes
```

Finally, add the hook to trigger Jenkins documentation construction (in this case, cutting/pasting the hook from another repo to get the link correct).

```
[git@jira git]$ tree dims-ad.git/hooks/
dims-ad.git/hooks/
+- post-receive -> /opt/git/bin/post-receive
+- post-receive-00logamqp -> /opt/git/bin/post-receive-00logamqp
+- post-receive-01email -> /opt/git/bin/post-receive-01email
+- post-receive-06jenkinsalldocs -> /opt/git/bin/post-receive-06jenkinsalldocs
```

```

0 directories, 4 files
[git@jira git]$ ln -s /opt/git/bin/post-receive-06jenkinsalldocs dims-asbuilt.git/
↪hooks/post-receive-06jenkinsalldocs
[git@jira git]$ tree dims-asbuilt.git/hooks/
dims-asbuilt.git/hooks/
+- post-receive -> /opt/git/bin/post-receive
+- post-receive-00logamqp -> /opt/git/bin/post-receive-00logamqp
+- post-receive-01email -> /opt/git/bin/post-receive-01email
+- post-receive-06jenkinsalldocs -> /opt/git/bin/post-receive-06jenkinsalldocs

0 directories, 4 files

```

Permanently Removing Files from a Git Repo

There are times when files exist in the repo (either active, or no longer active, but still included in past commits) that you want to permanently remove from the repo. Simply doing `git rm file` is not good enough. A common reason for doing this is if someone decided to commit many large binary archive files (e.g., some source packages, operating system installation ISOs, etc).

Danger: Realize that if you are trying to permanently remove secrets, such as passwords or encryption private keys, even doing these steps is not enough. Right now, go read the GitHub [Remove sensitive data](#) and its warning before going any further.

- [How to delete files permanently from your local and remote git repositories](#), by Anoopjohn, February 20, 2014
- GitHub [aaronzirbes/shrink-git-repo.sh](#) (“This script will help you remove large files from your git repo history and shrink the size of your repository.”)
- [How to Shrink a Git Repository](#), by Steve Lorek, May 11, 2012

The page [How to Shrink a Git Repository](#) was used successfully to perform cleanup of a large number of archives that were committed to the `ansible-playbooks` repo. The string `filename` needed to be substituted with the paths of the files to delete, which were identified by the script `git-find-largest` and edited with `vi` and `awk` to strip out just the paths. The following command was then used on the list:

```

for f in $(cat largest.txt); do \
    git filter-branch --tag-name-filter cat \
        --index-filter "git rm -r --cached --ignore-unmatch $f" \
        --prune-empty -f -- --all; \
done

```

After that, the steps to clear the cache, do garbage collection and pruning, etc. were followed.

Documenting DIMS Components

This chapter covers [Sphinx](#) and [ReStructured Text \(reST\)](#), and how they are used with [ReadTheDocs](#) (a hosted documentation site) and [GitHub](#) (a hosted Git source repository site) to document open source project repositories. It includes specifics of how Sphinx is used for documentation within the DIMS project.

Required Background Reading

Before trying to use Sphinx, it is important to understand how it works and what basic things you can do with it to produce organized and structured documentation that includes things like headings, tables, figures, images, links, cross-references to labelled items, and callout notes.

Start by taking less than five minutes and reading **all** of the *very short* [Sphinx Style Guide](#). It will give you some insight into high-level concepts of Sphinx and reST.

Next, spend another 10-15 minutes and read through **all** of the slightly longer [Documenting Your Project Using Sphinx](#) document to see the full range of markup and directives supported by reST.

A short tutorial that includes an example is IBM's [Easy and beautiful documentation with Sphinx](#).

A much longer (2+hours when delivered live) [Sphinx Tutorial v0.1](#) by Brandon Rhodes from PyCon 2013 walks through the full range of tasks necessary to document a Python code project.

Lastly, read [Problems with StructuredText](#) to learn about limitations in reST and some ways to deal with them.

Why Sphinx?

Just to illustrate how widely Sphinx is used in the open source community, here is a list of project repos in Dave Dittrich's [\\$GIT](#) directory that use Sphinx (by virtue of their containing a Sphinx configuration file `conf.py` under a documentation directory):

```
[dittrich@localhost git]$ find . -name conf.py
./ansible/docsite/conf.py
./celery/docs/conf.py
```

```
./crits/documentation/src/conf.py
./cuckoo/docs/book/src/conf.py
./CybOXProject/python-cybox/docs/conf.py
./elasticsearch-dsl-py/docs/conf.py
./MAECProject/python-maec/docs/conf.py
./MozDef/docs/source/conf.py
./pika/docs/conf.py
./pyxb/doc/conf.py
./redis-py/docs/conf.py
./robotframework/doc/api/conf.py
./sphinx_rtd_theme/demo_docs/source/conf.py
./STIXProject/python-stix/docs/conf.py
./TAXIIProject/libtaxii/docs/conf.py
./thug/doc/source/conf.py
```

Sphinx, since it is a Python project, is effectively programmable and highly configurable and flexible. You can do parameterized creation of documents to make them unique to a site using an open source software product, can exercise tests in code, can produce HTML and LaTeX-derived PDF, all from the same source documentation files. That is just the start. Sphinx also produces search indexes, dynamic tables of contents, forward and back buttons in HTML pages, and many other helpful features for documenting a project. Because it effectively compiles the documentation, things like unit tests, functional tests, software version descriptions, insertion of [Graphviz](#) directed and undirected graphs to illustrate relationships between system components... The list goes on.

Manually Initiating a docs directory with `sphinx-quickstart`

The program `sphinx-quickstart` can be used to initiate a Sphinx document directory. It is important to understand the ramifications of the first three questions in the context of how other Sphinx tools (e.g., `sphinx-autobuild`) work. Use of `sphinx-autobuild` is covered later. Here are the first two questions you are faced with after running `sphinx-quickstart` and what results from the choice.

```
[dittrich@localhost tmp]$ sphinx-quickstart
Welcome to the Sphinx 1.2.3 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
> Root path for the documentation [.]:

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n]: y
```

Separated source and build directories

Answering `y` to the second question (as shown above) results a having separate `source` and `build` directories, with the following structure:

```
.
+- Makefile
+- build
+- source
```

```
+-- _static
+-- _templates
+-- conf.py
+-- index.rst
```

4 directories, 3 files

When you initiate a build with `make html`, here is what the resulting directory contents will include:

```
.
+-- Makefile
+-- build
|   +- doctrees
|   |   +- environment.pickle
|   |   +- index.doctree
|   +- html
|       +- _sources
|       |   +- index.txt
|       +- _static
|       |   +- ajax-loader.gif
|       |   +- basic.css
|       |   +- comment-bright.png
|       |   +- comment-close.png
|       |   +- comment.png
|       |   +- default.css
|       |   +- doctools.js
|       |   +- down-pressed.png
|       |   +- down.png
|       |   +- file.png
|       |   +- jquery.js
|       |   +- minus.png
|       |   +- plus.png
|       |   +- pygments.css
|       |   +- searchtools.js
|       |   +- sidebar.js
|       |   +- underscore.js
|       |   +- up-pressed.png
|       |   +- up.png
|       |   +- websupport.js
|       +- genindex.html
|       +- index.html
|       +- objects.inv
|       +- search.html
|       +- searchindex.js
+-- source
    +- _static
    +- _templates
    +- conf.py
    +- index.rst
```

8 directories, 31 files

Note: Notice how the `build/` directory now contains subdirectories `html/` and `doctrees/` directories. There were no files created or changed in `source/` directory by the `make` operation.

Warning: You should answer **y** to the second question. DIMS project repositories should have **separated** source/ and build/ directories.

Mixed source and build

Had the second and third questions above been answered with a **n**, this is what the resulting directory structure would look like:

```
.
|- Makefile
|- _build
|- _static
|- _templates
|- conf.py
+-- index.rst

3 directories, 3 files
```

Notice the `conf.py` and `index.rst` files are located in the same directory root as `_build`. When you build this document with `make html`, the resulting directory structure now looks like this:

```
.
+-- Makefile
+-- _build
|   +- doctrees
|   |   +- environment.pickle
|   |   +- index.doctree
|   +- html
|       +- _sources
|       |   +- index.txt
|       +- _static
|       |   +- ajax-loader.gif
|       |   +- basic.css
|       |   +- comment-bright.png
|       |   +- comment-close.png
|       |   +- comment.png
|       |   +- default.css
|       |   +- doctools.js
|       |   +- down-pressed.png
|       |   +- down.png
|       |   +- file.png
|       |   +- jquery.js
|       |   +- minus.png
|       |   +- plus.png
|       |   +- pygments.css
|       |   +- searchtools.js
|       |   +- sidebar.js
|       |   +- underscore.js
|       |   +- up-pressed.png
|       |   +- up.png
|       |   +- websupport.js
|   +- genindex.html
|   +- index.html
|   +- objects.inv
|   +- search.html
|   +- searchindex.js
```



```
+ _static
+ _templates
+ conf.py
+ index.rst

7 directories, 31 files
```

Note: In this second example, the source files `index.rst` and the `conf.py` file are at the same directory level as the `_build/` directory (and all of its contents). Doing a `make html` or `make latexpdf` both cause the source directory `.` to change, because new files and directories were created within the `.` directory.

The `sphinx-quickstart` program gives you an option of separating the source directory from other directories. When this option is chosen, the result is a directory structure that has the `Makefile` at the top level with a `build` and source directory at the same directory level, which looks like this:

```
.
|- Makefile
|- build
+- source
    +- README.rst
    +- _static
    +- _templates
    +- conf.py
    +- developing.rst
    +- index.rst
    +- intro.rst
    +- license.rst
    +- quickstart.rst

4 directories, 8 files
```

Building Sphinx Documentation

You can build HTML manually with the `Makefile`, build PDF output with the `Makefile`, or automatically build HTML whenever files change on disk using `sphinx-autobuild`.

When you are ready to try building your documentation, start with manually building HTML output (which you can test locally with a browser). Once you understand how building HTML works, and know what to look for in terms of error messages and warnings, you will find it is faster and easier to create Sphinx documents using `sphinx-autobuild` and a browser in a second window.

Manually Building HTML

The most simple way to render Sphinx documents is to use the `Makefile` created by `sphinx-quickstart` using `make` as shown here:

```
[dittrich@localhost docs (dev)]$ make html
sphinx-build -b html -d build/doctrees   source build/html
Making output directory...
Running Sphinx v1.2.3
loading pickled environment... not yet created
loading intersphinx inventory from http://docs.python.org/objects.inv...
```

```
building [html]: targets for 8 source files that are out of date
updating environment: 8 added, 0 changed, 0 removed
reading sources... [ 12%] README
reading sources... [ 25%] continuousintegration
reading sources... [ 37%] deployconfigure
reading sources... [ 50%] developing
reading sources... [ 62%] documentation
reading sources... [ 75%] index
reading sources... [ 87%] introduction
reading sources... [100%] quickstart

looking for now-outdated files... none found
pickling environment... done
checking consistency... /Users/dittrich/git/dims-ci-utils/docs/source/README.rst::
↳WARNING: document isn't included in any toctree
done
preparing documents... done
writing output... [ 12%] README
writing output... [ 25%] continuousintegration
writing output... [ 37%] deployconfigure
writing output... [ 50%] developing
writing output... [ 62%] documentation
writing output... [ 75%] index
writing output... [ 87%] introduction
writing output... [100%] quickstart

writing additional files... genindex search
copying images... [100%] images/DD_home_page_small.jpg

copying downloadable files... [100%] /Users/dittrich/git/dims-ci-utils/docs/source/
↳images/DD_home_page.png

copying static files... done
copying extra files... done
dumping search index... done
dumping object inventory... done
build succeeded, 1 warning.

Build finished. The HTML pages are in build/html.
```

You can now load the page with a browser:

```
[dittrich@localhost docs (dev)]$ open -a Opera.app build/html/index.html
```

Manually Building PDF using LaTeX

Now, render the same document as a PDF file using LaTeX:

```
[dittrich@localhost docs (dev)]$ make latexpdf
sphinx-build -b latex -d build/doctrees source build/latex
Making output directory...
Running Sphinx v1.2.3
loading pickled environment... done
building [latex]: all documents
updating environment: 0 added, 0 changed, 0 removed
looking for now-outdated files... none found
```

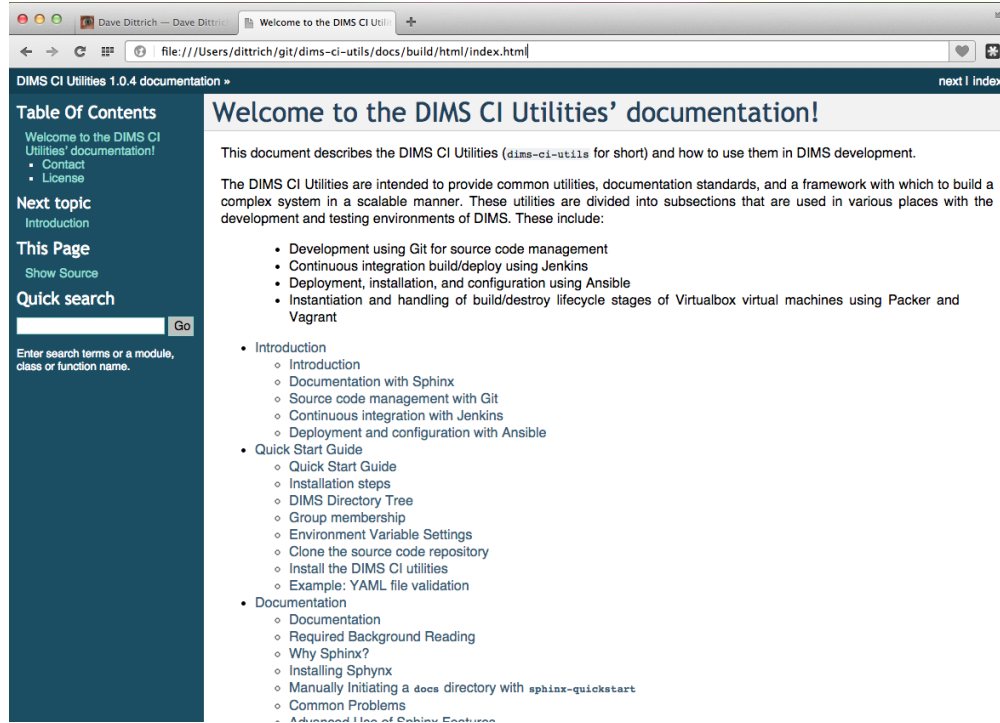


Fig. 5.1: This documentation, rendered on a Mac using Opera.

```

processing DIMSCIUtilities.tex... index introduction quickstart documentation_
↳developing continuousintegration deployconfigure
resolving references...
writing... done
copying images... dims-ci-utils-doc.png DD_home_page_small.jpg
copying TeX support files...
done
build succeeded.
Running LaTeX files through pdflatex...
/Applications/Xcode.app/Contents/Developer/usr/bin/make -C build/latex all-pdf
pdflatex 'DIMSCIUtilities.tex'
This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014/MacPorts 2014_4)
↳(preloaded format=pdflatex)
restricted \writel8 enabled.
entering extended mode
(./DIMSCIUtilities.tex
LaTeX2e <2014/05/01>
Babel <3.9k> and hyphenation patterns for 43 languages loaded.
(./sphinxmanual.cls
Document Class: sphinxmanual 2009/06/02 Document class (Sphinx manual)
(/opt/local/share/texmf-texlive/tex/latex/base/report.cls
Document Class: report 2007/10/19 v1.4h Standard LaTeX document class
(/opt/local/share/texmf-texlive/tex/latex/base/size10.clo))
(/opt/local/share/texmf-texlive/tex/latex/base/inputenc.sty

[ ...pages of output removed... ]

[25] [26]
Chapter 6.

```

```
[27] [28]
Chapter 7.
[29] [30]
Chapter 8.
(./DIMSCIUtilities.ind) [31] (./DIMSCIUtilities.aux)
(see the transcript file for additional information){/opt/local/share/texmf-tex
live/fonts/enc/dvips/base/8r.enc}</opt/local/share/texmf-texlive/fonts/type1/ur
w/courier/ucrb8a.pfb></opt/local/share/texmf-texlive/fonts/type1/urw/courier/uc
rr8a.pfb></opt/local/share/texmf-texlive/fonts/type1/urw/courier/ucrr8a.pfb></
opt/local/share/texmf-texlive/fonts/type1/urw/helvetica/uhvb8a.pfb></opt/local/s
hare/texmf-texlive/fonts/type1/urw/helvetica/uhvbo8a.pfb></opt/local/share/texmf
-texlive/fonts/type1/urw/times/utmb8a.pfb></opt/local/share/texmf-texlive/fonts
/type1/urw/times/utmr8a.pfb></opt/local/share/texmf-texlive/fonts/type1/urw/tim
es/utmri8a.pfb>
Output written on DIMSCIUtilities.pdf (35 pages, 381656 bytes).
Transcript written on DIMSCIUtilities.log.
pdflatex finished; the PDF files are in build/latex.
```

Now open the PDF file (this example uses Mac OS X Preview.app, but you can also use evince on some Linux systems):

```
[dittrich@localhost docs (dev)]$ open build/latex/DIMSCIUtilities.pdf
```

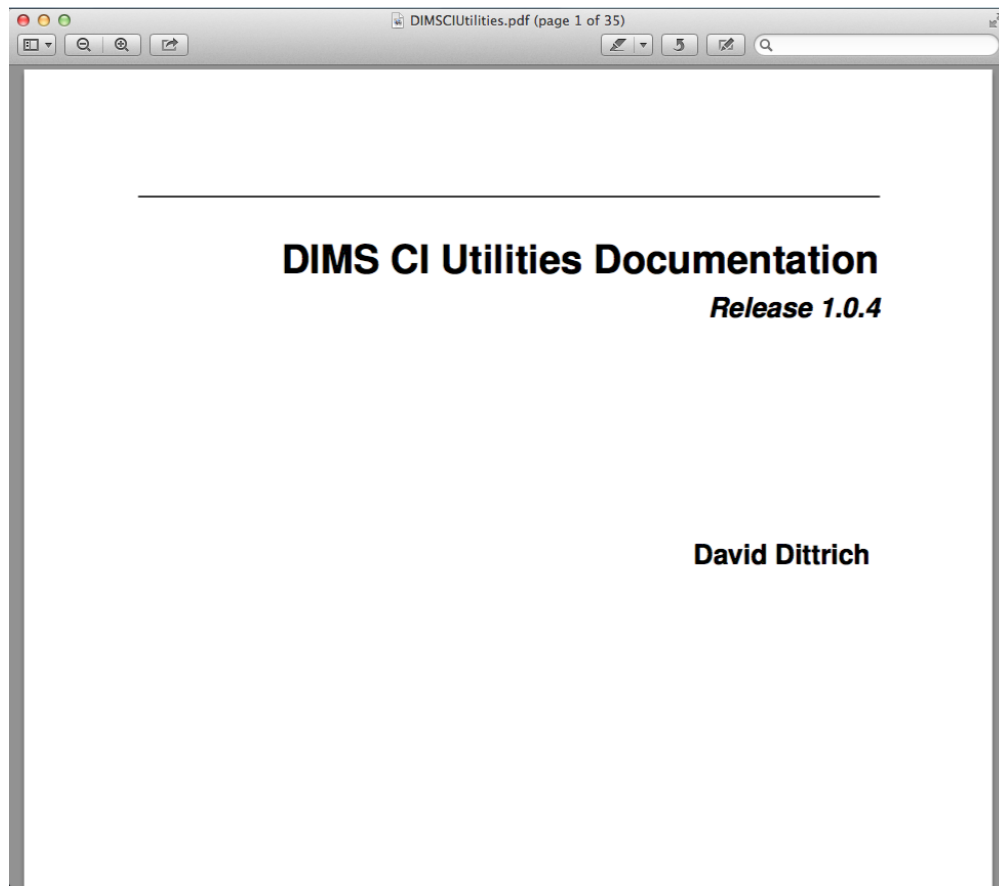


Fig. 5.2: This documentation, rendered using LaTeX on a Mac, viewed with Preview.

Automatically building HTML

Sphinx has a program called `sphinx-autobuild` that can monitor a directory for any file changes in that directory and below, re-building the document immediately upon detecting changes. When used to build HTML content, it makes the pages available on a local TCP port using a simple HTTP service (just like if the docs were put up on GitHub, readthedocs, etc.)

Note: You may need to install `sphinx-autobuild` using `pip` separately. Refer to section `installingsphinx`.

Here is where the importance of splitting the `source/` directory from `build/` directory becomes evident.

Invoke `sphinx-autobuild` from the command line in a separate terminal window, so you can watch the output for error messages. By default, `sphinx-autobuild` listens on 8000/tcp. (This can be changed with the `-p` flag on the command line). After starting `sphinx-autobuild` you then enter the URL that is produced (in this case, the URL is `http://127.0.0.1:8000`). Now edit files in another terminal or editor application window.

```
[dittrich@localhost docs (dev)]$ sphinx-autobuild --ignore '*.swp' source build/html
Serving on http://127.0.0.1:8000
[I 150105 18:50:45 handlers:109] Browser Connected: http://127.0.0.1:8000/
→documentation.html
[I 150105 18:50:45 handlers:118] Start watching changes
[I 150105 18:50:48 handlers:74] Reload 1 waiters: None
[I 150105 18:50:48 web:1811] 200 GET /documentation.html (127.0.0.1) 16.57ms
[I 150105 18:50:48 web:1811] 304 GET /livereload.js (127.0.0.1) 1.08ms
[I 150105 18:50:48 web:1811] 200 GET /_static/pygments.css (127.0.0.1) 0.83ms
[I 150105 18:50:48 web:1811] 200 GET /_static/default.css (127.0.0.1) 0.62ms
[I 150105 18:50:48 web:1811] 200 GET /_static/jquery.js (127.0.0.1) 1.24ms
[I 150105 18:50:48 web:1811] 200 GET /_static/underscore.js (127.0.0.1) 1.09ms
[I 150105 18:50:48 web:1811] 200 GET /_static/doctools.js (127.0.0.1) 0.68ms
[I 150105 18:50:48 web:1811] 200 GET /_images/DD_home_page_small.jpg (127.0.0.1) 0.
→86ms
[I 150105 18:50:48 web:1811] 200 GET /_static/basic.css (127.0.0.1) 0.46ms
[I 150105 18:50:48 web:1811] 200 GET /_images/dims-ci-utils-doc-html.png (127.0.0.1)
→1.59ms
[I 150105 18:50:48 web:1811] 200 GET /_images/dims-ci-utils-doc-pdf.png (127.0.0.1) 0.
→72ms
[I 150105 18:50:48 handlers:109] Browser Connected: http://127.0.0.1:8000/
→documentation.html

+----- source/documentation.rst changed -----
| Running Sphinx v1.2.3
| loading pickled environment... not yet created
| No builder selected, using default: html
| loading intersphinx inventory from http://docs.python.org/objects.inv...
| building [html]: targets for 8 source files that are out of date
| updating environment: 8 added, 0 changed, 0 removed
| reading sources... [ 12%] README
| reading sources... [ 25%] continuousintegration
| reading sources... [ 37%] deployconfigure
| reading sources... [ 50%] developing
| reading sources... [ 62%] documentation
| reading sources... [ 75%] index
| reading sources... [ 87%] introduction
| reading sources... [100%] quickstart
/Users/dittrich/git/dims-ci-utils/docs/source/documentation.rst:281: WARNING: Literal
→block ends without a blank line; unexpected unindent.
/Users/dittrich/git/dims-ci-utils/docs/source/documentation.rst:519: WARNING: Literal
→block ends without a blank line; unexpected unindent.
```

```
|
| looking for now-outdated files... none found
| pickling environment... done
/Users/dittrich/git/dims-ci-utils/docs/source/README.rst:: WARNING: document isn't
↳included in any toctree
| checking consistency... done
| preparing documents... done
| writing output... [ 12%] README
| writing output... [ 25%] continuousintegration
| writing output... [ 37%] deployconfigure
| writing output... [ 50%] developing
| writing output... [ 62%] documentation
| writing output... [ 75%] index
| writing output... [ 87%] introduction
| writing output... [100%] quickstart
|
| writing additional files... genindex search
| copying images... [ 33%] dims-ci-utils-doc-pdf.png
| copying images... [ 66%] DD_home_page_small.jpg
| copying images... [100%] dims-ci-utils-doc-html.png
|
| copying downloadable files... [100%] /Users/dittrich/git/dims-ci-utils/docs/source/
↳images/DD_home_page.png
|
| copying static files... done
| copying extra files... done
| dumping search index... done
| dumping object inventory... done
| build succeeded, 3 warnings.
+-----

+----- source/documentation.rst changed -----
| Running Sphinx v1.2.3
| loading pickled environment... done
| No builder selected, using default: html
| building [html]: targets for 0 source files that are out of date
| updating environment: 0 added, 0 changed, 0 removed
| looking for now-outdated files... none found
| no targets are out of date.
+-----

[I 150105 18:51:17 handlers:74] Reload 1 waiters: None
[I 150105 18:51:17 web:1811] 200 GET /documentation.html (127.0.0.1) 1.70ms
[I 150105 18:51:17 web:1811] 200 GET /_static/default.css (127.0.0.1) 0.70ms
[I 150105 18:51:17 web:1811] 200 GET /_static/doctools.js (127.0.0.1) 0.76ms
[I 150105 18:51:17 web:1811] 200 GET /_static/underscore.js (127.0.0.1) 0.88ms
[I 150105 18:51:17 web:1811] 200 GET /_static/jquery.js (127.0.0.1) 1.26ms
[I 150105 18:51:17 web:1811] 200 GET /_static/pygments.css (127.0.0.1) 0.71ms
[I 150105 18:51:17 web:1811] 304 GET /livereload.js (127.0.0.1) 0.83ms
[I 150105 18:51:17 web:1811] 200 GET /_images/DD_home_page_small.jpg (127.0.0.1) 1.
↳04ms
[I 150105 18:51:17 web:1811] 200 GET /_static/basic.css (127.0.0.1) 0.54ms
[I 150105 18:51:17 web:1811] 200 GET /_images/dims-ci-utils-doc-html.png (127.0.0.1)
↳1.86ms
[I 150105 18:51:17 web:1811] 200 GET /_images/dims-ci-utils-doc-pdf.png (127.0.0.1) 0.
↳96ms
[I 150105 18:51:17 handlers:109] Browser Connected: http://127.0.0.1:8000/
↳documentation.html
```

Every time you change a file, `sphinx-autobuild` will rebuild it and your browser will be informed that it needs to reload the page so you can immediately see the results. This helps in developing Sphinx documentation quickly, as all you need to do is edit files and watch for error messages in the `sphinx-autobuild` window and see if the browser page shows what you want it to show.

Warning: The above example uses `--ignore '*.swp'` to avoid temporary swap files created by the `vim` editor. If you use an editor that creates temporary files using a different file extension, you should use that name instead. Otherwise, every time you open a file with the editor it will appear to `sphinx-autobuild` as though a source file changed and it will regenerate the document.

Warning: If you restart the `sphinx-autobuild` process, you will need to reconnect the browser to the `sphinx-autobuild` listening port, otherwise the browser will stop updating the page automatically at the end of each automatic build. Refreshing the page can fix this.

If you start the browser and attempt to re-open a previously used URL *before* you start `sphinx-autobuild`, you may experience a similar problem. Try to use `touch` to update a file, or edit a file and force a write operation. Either of these will trigger a rebuild and refresh of the browser, which should then keep it in sync.

The example above produces a **lot** of output in the `sphinx-autobuild` terminal output, which in practice makes it a little harder to see the error messages. To decrease the amount of output, you may want to add the `-q` flag (see also `sphinx-build -h` for how to control the underlying build process, and `sphinx-autobuild --help` for more autobuild options).

```
[dittrich@localhost docs (dev)]$ sphinx-autobuild -q --ignore '*.swp' source build/
↪html
```

Warning: By default, `sphinx-autobuild` will attempt to bind to port 8000/tcp. If that port is in use by another instance of `sphinx-autobuild`, you will get an error message. Use the `-p` flag to change the listening port number to something else (e.g., `-p 8001`).

Fixing errors

If there are any problems, Sphinx will call them out with warnings. Pay attention to the build output.

```
rm -rf build/*
sphinx-build -b html -d build/doctrees    source build/html
Making output directory...
Running Sphinx v1.2.3
loading pickled environment... not yet created
loading intersphinx inventory from http://docs.python.org/objects.inv...
building [html]: targets for 7 source files that are out of date
updating environment: 7 added, 0 changed, 0 removed
reading sources... [ 14%] README
reading sources... [ 28%] advanced
reading sources... [ 42%] developing
reading sources... [ 57%] index
reading sources... [ 71%] intro
reading sources... [ 85%] license
```

```
reading sources... [100%] quickstart

/Users/dittrich/git/dims-ci-utils/docs/source/intro.rst:26: WARNING: Inline literal
↳start-string without end-string.
/Users/dittrich/git/dims-ci-utils/docs/source/intro.rst:95: WARNING: Literal block
↳ends without a blank line; unexpected unindent.
looking for now-outdated files... none found
pickling environment... done
checking consistency...
/Users/dittrich/git/dims-ci-utils/docs/source/README.rst:: WARNING: document isn't
↳included in any toctree
/Users/dittrich/git/dims-ci-utils/docs/source/advanced.rst:: WARNING: document isn't
↳included in any toctree
/Users/dittrich/git/dims-ci-utils/docs/source/license.rst:: WARNING: document isn't
↳included in any toctree
done
preparing documents... done
writing output... [ 14%] README
writing output... [ 28%] advanced
writing output... [ 42%] developing
writing output... [ 57%] index
writing output... [ 71%] intro
writing output... [ 85%] license
writing output... [100%] quickstart

writing additional files... genindex search
copying static files... done
copying extra files... done
dumping search index... done
dumping object inventory... done
build succeeded, 23 warnings.

Build finished. The HTML pages are in build/html.
```

Typographic errors

Both of the errors seen in this first example above are simple typographical errors in the `intro.rst` file.

The first one, as it says, involves an improper literal on line 25:

```
25  A much longer (2+hours when delivered live) ``Sphinx Tutorial v0.1`` by Brandon
26  Rhodes from PyCon 2013 walks through the full range of tasks necessary to
27  document a Python code project.
```

Here is the context for the second error message, regarding line 95:

```
73  Manually Initiating a ``docs`` directory with ``sphinx-quickstart``
74  -----
75
76  The ``sphinx-quickstart`` program gives you an option of separating the source
77  directory from other directories. The result is a directory structure that
78  looks like this: ::
79
80      .
81      +- Makefile
82      +- build
83      +- source
```



```

84         +- README.rst
85         +- _static
86         +- _templates
87         +- conf.py
88         +- developing.rst
89         +- index.rst
90         +- intro.rst
91         +- license.rst
92         +- quickstart.rst
93
94     4 directories, 8 files
95 ..
96

```

As you can see, there is no blank line before the end of the literal block that ends on line 94 and before the reST comment tag (..) on line 95 (the one identified in the error message).

This is a simple error, but it happens quite frequently when inserting literal text examples. If need be, go back and re-read [Sphinx Style Guide](#) and [Documenting Your Project Using Sphinx](#) every now and then when you are starting out to get a refresher, and also have a browser window up with the [The reStructuredText_ Cheat Sheet: Syntax Reminders](#) or [Quick reStructuredText](#) quick reference guide to help while writing reST documents.

Link errors

A more subtle problem that comes up frequently when creating links to reference material in Sphinx documents is this error:

```

/Users/dittrich/git/dims-ci-utils/docs/source/intro.rst:274: ERROR: Unknown
target name: "the restructuredtext_ cheat sheet: syntax reminders".

```

See if you can spot the reason why by looking very closely at lines 274 and 323 before reading the explanation that follows:

```

...
273     are starting out to get a refresher, and also have a browser window
274     up with the `The reStructuredText_ Cheat Sheet: Syntax Reminders`_ or
275     `Quick reStructuredText`_ quick reference guide to help while
276     writing reST documents.
...
321 .. _Sphinx Tutorial v0.1: http://brandons-sphinx-tutorial.readthedocs.org/en/v0.
↪1/
323 .. _The reStructuredText_ Cheat Sheet: Syntax Reminders: http://docutils.
↪sourceforge.net/docs/user/rst/cheatsheet.txt
324 .. _Quick reStructuredText: http://docutils.sourceforge.net/docs/user/rst/
↪quickref.html

```

Unlike the links on lines 321 and 324, the target string specified on line 323 has *two* colons in it. This causes Sphinx to parse the line incorrectly (which in turn causes the Unknown target name error to be triggered). The error is not really on line 274, but is actually on line 323! It just presents itself as a missing target error on line 274. The solution is to make sure that all colons in targets for links are escaped, **except** the one before the URL, like this:

```

323 .. _The reStructuredText_ Cheat Sheet\: Syntax Reminders: http://docutils.
↪sourceforge.net/docs/user/rst/cheatsheet.txt

```

LaTeX image errors

You may get errors rendering LaTeX PDF documents that include image files. Such an error may look like this:

```
[dittrich@localhost docs (feature/docs)]$ make latexpdf
sphinx-build -b latex -d build/doctrees source build/latex
Running Sphinx v1.2.3

...

Running LaTeX files through pdflatex...
/Applications/Xcode.app/Contents/Developer/usr/bin/make -C build/latex all-pdf
pdflatex 'DIMSCIUtilities.tex'
This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014/MacPorts 2014_4)
↪ (preloaded format=pdflatex)

...

Chapter 1.
[3] [4] (/opt/local/share/texmf-texlive/tex/latex/psnfss/tslpcr.fd) [5]

pdfTeX warning: pdflatex: arithmetic: number too big
! Dimension too large.
<argument> \ht \@tempboxa

1.348 ...=0.800\linewidth]{images/DD_home_page_small.png}

? q
OK, entering \batchmodemake[1]: *** [DIMSCIUtilities.pdf] Error 1
make: *** [latexpdf] Error 2
```

The solution to this is to use `mogrify -density 90 DD_home_page_small.png` to fix the image resolution metadata in the PNG file.

LaTeX Unicode rendering errors

Another error message that could occur when rendering the kind of text in the code-block seen in Section *Typographic errors* relates to the Unicode characters produced by the `tree` program to show the indentation levels.

Here is an error message (with the specific lines highlighted) that can show up in a Jenkins build process FAILURE message:

```
1 Running LaTeX files through pdflatex...
2 make -C build/latex all-pdf
3 make[1]: Entering directory `/var/lib/jenkins/jobs/dims-docs-deploy/workspace/ansible-
↪playbooks/docs/build/latex'
4 pdflatex 'AnsiblePlaybooksRepository.tex'
5 This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
6 entering extended mode
7 (./AnsiblePlaybooksRepository.tex
8
9 ...
10
11 Underfull \hbox (badness 10000) in paragraph at lines 819--822
12 []\Tl/ptm/m/n/10 While it is not re-quired to in-stall dims-ci-utils, you prob-
13 a-bly will want to run the play-book
14 [11] [12] [13]
```

```

15
16 ! Package inputenc Error: Unicode char \u8:a"œ not set up for use with LaTeX.
17
18 See the inputenc package documentation for explanation.
19 Type H <return> for immediate help.
20 ...
21
22 l.1040 â"œ-- defaults
23
24 ?
25 ! Emergency stop.
26 ...
27
28 l.1040 â"œ-- defaults
29
30 ! ==> Fatal error occurred, no output PDF file produced!
31 Transcript written on AnsiblePlaybooksRepository.log.
32 make[1]: *** [AnsiblePlaybooksRepository.pdf] Error 1
33 make[1]: Leaving directory `/var/lib/jenkins/jobs/dims-docs-deploy/workspace/ansible-
↳playbooks/docs/build/latex'
34 make: *** [latexpdf] Error 2
35 Build step 'Custom Python Builder' marked build as failure
36 Warning: you have no plugins providing access control for builds, so falling back to
↳legacy behavior of permitting any downstream builds to be triggered
37 Finished: FAILURE

```

Here is the specific block of text that triggered the rendering error message:

```

.. code-block:: bash

- defaults
  - main.yml
- files
  - base-requirements.txt
  - debian-virtualenv-prereqs.sh
  - dimsenv-requirements.txt
  - macos-virtualenv-prereqs.sh
- meta
  - main.yml
- tasks
  - main.yml
  - post_tasks.yml -> ../../../../dims/post_tasks.yml
  - pre_tasks.yml -> ../../../../dims/pre_tasks.yml
- templates
  - bashrc.dims.virtualenv.j2
  - bulddimsenvmod.sh.j2

..

```

The problem is that the long-dash character is not defined to LaTeX. This is done in the Sphinx `conf.py` file, and all DIMS documents should include these definitions because we frequently embed output of `tree`, which uses Unicode characters for line drawing. (Not all do, which causes random failures when adding text to Sphinx documents.)

```

latex_elements = {
    ...
    # Additional stuff for the LaTeX preamble.
    #
    # The following comes from

```

```
# https://github.com/rtfd/readthedocs.org/issues/416
#
'preamble': """.join((
    '\DeclareUnicodeCharacter{00A0}{ }',      # NO-BREAK SPACE
    '\DeclareUnicodeCharacter{2014}{\dash}',  # LONG DASH
    '\DeclareUnicodeCharacter{251C}{+}',      # BOX DRAWINGS LIGHT VERTICAL AND RIGHT
    '\DeclareUnicodeCharacter{2514}{+}',      # BOX DRAWINGS LIGHT UP AND RIGHT
)),
}
```

Note: See <http://tex.stackexchange.com/questions/34604/entering-unicode-characters-in-latex>

“LaTeX is not a TTY” errors

Another variation of errors during LaTeX rendering presents itself similarly to the previous error, but the problem is due to inability to map a Unicode character to a LaTeX macro: the problem is due to directly (or indirectly) sending output saved from Unix command line programs that do fancy things like coloring characters, etc, using ANSI escape sequences. While a terminal program that uses the Unix TTY subsystem may handle the ANSI escape sequences, and HTML renderers may know how to handle the ANSI escape sequences, LaTeX does not. Here is an example of this problem, excerpted from a Jenkins build job email message:

```
1  Started by user anonymous
2  [EnvInject] - Loading node environment variables.
3  Building in workspace /var/lib/jenkins/jobs/dims-docs-deploy/workspace
4
5  Deleting project workspace... done
6
7  [workspace] $ /bin/bash -xe /tmp/shiningpanda5607640542889107840.sh
8  + jenkins.logmon
9  [workspace] $ /bin/bash -xe /tmp/shiningpanda5535708223044870299.sh
10 + jenkins.dims-docs-deploy
11 [+++] jenkins.dims-docs-deploy: Deploying documentation
12 [+++] jenkins.dims-docs-deploy: Get global vars from jenkins.dims-defaults.
13 [+++] jenkins.dims-defaults Default variables
14 [+++]   PLAYBOOKSREPO=ansible-playbooks
15 [+++]   INVENTORYREPO=ansible-inventory
16 [+++]   GITURLPREFIX=git@git.prisem.washington.edu:/opt/git/
17 [+++]   MASTERBRANCH=master
18 [+++]   DEVBRANCH=develop
19 [+++]   DEVHOSTS=development
20 [+++]   MASTERHOSTS=production
21 [+++]   DEFAULTHOSTFILE=development
22 [+++]   DEFAULTANSIBLEBRANCH=develop
23 [+++]   DEFAULTINVENTORYBRANCH=develop
24 [+++]   DEFAULTREMOTEUSER=ansible
25
26 ...
27 ! Package inputenc Error: Keyboard character used is undefined
28 (inputenc)                in inputencoding `utf8'.
29
30 See the inputenc package documentation for explanation.
31 Type H <return> for immediate help.
32 ...
33
```

```

34 1.5790 ...dl{}GIT/dims\PYGZhy{}dockerfiles/configu
35
36 ! ==> Fatal error occurred, no output PDF file produced!
37 Transcript written on UsingDockerinDIMS.log.
38 make[1]: *** [UsingDockerinDIMS.pdf] Error 1
39 make[1]: Leaving directory `/var/lib/jenkins/jobs/dims-docs-deploy/workspace/dims-
↳ dockerfiles/docs/build/latex'
40 make: *** [latexpdf] Error 2
41 Build step 'Custom Python Builder' marked build as failure
42 Warning: you have no plugins providing access control for builds, so falling back to
↳ legacy behavior of permitting any downstream builds to be triggered
43 Finished: FAILURE

```

To find the line in question (5790, in this case, called out in output line 34 above), manually trigger a LaTeX PDF build from the Sphinx document and then look for the LaTeX source file that corresponds with the PDF file name (seen in output line 38 above) in the build/latex subdirectory (in this case, it would be \$GIT/dims-dockerfiles/docs/build/latex/UsingDockerinDIMS.tex) to find the character that causes the error:

```

1  [dimsenv] ~/dims/git/dims-dockerfiles/docs (develop) $ make latexpdf
2
3  ...
4
5  ! Package inputenc Error: Keyboard character used is undefined
6  (inputenc)                  in inputencoding `utf8'.
7
8  See the inputenc package documentation for explanation.
9  Type H <return> for immediate help.
10 ...
11
12 1.5789 ...dl{}GIT/dims\PYGZhy{}dockerfiles/configu
13
14 ? ^Cmake[1]: *** Deleting file `UsingDockerinDIMS.pdf'
15 ^Z
16 [1]+  Stopped                  make latexpdf
17 [dimsenv] ~/dims/git/dims-dockerfiles/docs (develop) $ kill -9 %1
18 [1]+  Killed: 9                make latexpdf
19 [dimsenv] ~/dims/git/dims-dockerfiles/docs (develop) $ pr -n build/latex/
↳ UsingDockerinDIMS.tex | less
20
21 ...
22
23 5780 * VPN \PYGZsq{}01\PYGZus{}uwapl\PYGZus{}dimsdev2\PYGZsq{} is running
24 5781 * VPN \PYGZsq{}02\PYGZus{}prsm\PYGZus{}dimsdev2\PYGZsq{} is running
25 5782 [+++] Sourcing /opt/dims/etc/bashrc.dims.d/bashrc.dims.virtualenv ...
26 5783 [+++] Activating DIMS virtual environment (dimsenv) [ansible\PYGZhy{}
↳ playbooks v1.2.93]
27 5784 [+++] (Create file /home/mboggess/.DIMS\PYGZus{}NO\PYGZus{}DIMSENV\PYGZus{}
↳ ACTIVATE to disable)
28 5785 [+++] Virtual environment \PYGZsq{}dimsenv\PYGZsq{} activated [ansible\PYGZhy
↳ {}playbooks v1.2.93]
29 5786 [+++] /opt/dims/bin/dims.install.dimscommands: won\PYGZsq{}t try to install
↳ scripts in /opt/dims
30 5787 [+++] Sourcing /opt/dims/etc/bashrc.dims.d/git\PYGZhy{}prompt.sh ...
31 5788 [+++] Sourcing /opt/dims/etc/bashrc.dims.d/hub.bash\PYGZus{}completion.sh ...
32 5789 ESC[1;34m[dimsenv]ESC[0m ESC[1;33mmboggess@dimsdev2:\PYGZti{}core\PYGZhy{}
↳ localESC[0m () \PYGZdl{} bash \PYGZdl{}GIT/dims\PYGZhy{}dockerfiles/configu
33 5790 rations/elasticsearch/setup\PYGZus{}cluster.sh
34 5791

```

```

35 5792 elasticsearch@.service 0\PYGZpc{} 0 0.0KB/s
→ \PYGZhy{}\PYGZhy{}:\PYGZhy{}\PYGZhy{} ETA
36 5793 elasticsearch@.service 100\PYGZpc{} 1680 1.6KB/s
→ 00:00
37 5794
38 5795 start\PYGZus{}elasticsearch\PYGZus{}cluster.sh 0\PYGZpc{}
→ 0 0.0KB/s \PYGZhy{}\PYGZhy{}:\PYGZhy{}\PYGZhy{} ETA
39 5796 start\PYGZus{}elasticsearch\PYGZus{}cluster.sh 100\PYGZpc{}
→ 75 0.1KB/s 00:00
40 5797 ESC[1;34m[dimsenv]ESC[0m ESC[1;33mboguess@dimsdev2:\PYGZti{}/core\PYGZhy{
→ localESC[0m () \PYGZdl{} vagrant ssh core\PYGZhy{}01 \PYGZhy{}\PYGZhy{} \PYGZhy{}A
41 5798 VM name: core\PYGZhy{}01 \PYGZhy{} IP: 172.17.8.101
42 5799 Last login: Wed Sep 9 13:50:22 2015 from 10.0.2.2
43 5800
44 5801 CoreESC[38;5;206mOESC[38;5;45mSESC[39m alpha (794.0.0)
45 5802 ESC]0;core@core\PYGZhy{}01:\PYGZti{}^GESc[?1034hESC[01;32mcore@core\PYGZhy{
→ 01ESC[01;34m \PYGZti{} \PYGZdl{}ESC[00m ls
46 5803 ESC[0mESC[01;34minstancesESC[0m start\PYGZus{}elasticsearch\PYGZus{}cluster.
→ sh ESC[01;34mstaticESC[0m ESC[01;34mtemplatesESC[0m
47 5804 ESC]0;core@core\PYGZhy{}01:\PYGZti{}^GESc[01;32mcore@core\PYGZhy{}01ESC[01;
→ 34m \PYGZti{} \PYGZdl{}ESC[00m bash start\PYGZus{}elasticsearch\PYGZus{}cluster.sh
48 5805 ESC]0;core@core\PYGZhy{}01:\PYGZti{}^GESc[01;32mcore@core\PYGZhy{}01ESC[01;
→ 34m \PYGZti{} \PYGZdl{}ESC[00m ESC[Ketcdctl cluster\PYGZhy{}hea
49 ...

```

Note: Pay close attention to the commands used to reproduce the error that Jenkins encountered from the command line. LaTeX, which is being invoked by Sphinx (a Python program that invokes `pdflatex` as a subprocess) has some problems getting the **CTRL-C** character (see line 14). To work around this, do the following:

1. Suspend the process with **CTRL-Z** (see line 15).
2. Identify the suspended job's number found within the square brackets on line 16: (`[1]+ Stopped ...`, in this case, job 1).
3. Use the `kill` command (see `man kill` and `man signal`) to send the `-9` (non-maskable interrupt) signal to the suspended job (see line 17).
4. Use `pr -n` to add line numbers to the file and pass the output to a pager like `less` to find the line number called out by LaTeX (see lines 19 and 32).

As can be seen in line 32 above, the escape sequence **ESC[1;34m** (set foreground color 'Blue': see [Bash Prompt HOWTO: Chapter 6. ANSI Escape Sequences: Colours and Cursor Movement](#)) is causing LaTeX to fail.

The moral of the story is, only send properly mapped Unicode and/or UTF-8/ASCII text to Sphinx, so that when it does not fail when it invokes LaTeX.

Note: You can strip ANSI escape sequences in many ways. Google "strip ANSI escape sequences" to find some. Another way to handle this is to disable colorizing, or cut/paste command output as simple text rather than capturing terminal output with programs like `script`.

Common Tasks

Creating figures with thumbnails with links to larger images

Dave Dittrich's home page has a section with images, which uses low-resolution version for the main page and keeps the high-resolution image in a separate `_download` directory to be used as targets in the captions of those thumbnails. Here is a partial directory listing:

```
[dittrich@localhost sphinx (master)]$ tree
.
+- Makefile
+- _build
+- _download
+- . . .
+- images
|   +- Black_Mamba_Vienna-small.jpg
|   +- Black_Mamba_Vienna.jpg
|   +- Climbing_Gym_Manchester-small.jpg
|   +- Climbing_Gym_Manchester.jpg
|   +- DCA_Sunset-small.jpg
|   +- DCA_Sunset.jpg
|   +- QR-code-security-QR-code.gif
|   +- QR-code-security-QR-code.png
|   +- Sagrada_Familia_Barcelona-small.jpg
|   +- Sagrada_Familia_Barcelona.jpg
|   +- Screen-Shot-2014-12-31-at-1.15.34-PM.png
|   +- Seattle_Sunset_1-small.jpg
|   +- Seattle_Sunset_1.jpg
|   +- Seattle_Sunset_2-small.jpg
|   +- Seattle_Sunset_2.jpg
|   +- T-Rex-Chicago-small.jpg
|   +- T-Rex-Chicago.jpg
|   +- UW-Memorial-Way-Northbound-small.jpg
|   +- UW-Memorial-Way-Northbound.jpg
|   +- WA_OR_Volcanoes-small.jpg
|   +- WA_OR_Volcanoes.jpg
|   +- . . .
|   +- weber_guy.png
+- images.rst
+- . . .
+- www.rst
```

Here is how the figure with link works:

```
.. figure:: images/DD_home_page_small.jpg
   :alt: Dave Dittrich's home page
   :width: 80%
   :align: center

   A screen shot of Dave Dittrich's home page.
   :download:`Full size image <images/DD_home_page.png>
```

Note: Mouse over the image and right click and you will get the small image. Mouse over the words **Full size image** in the caption and right click and you get the... well, yes... full size image.

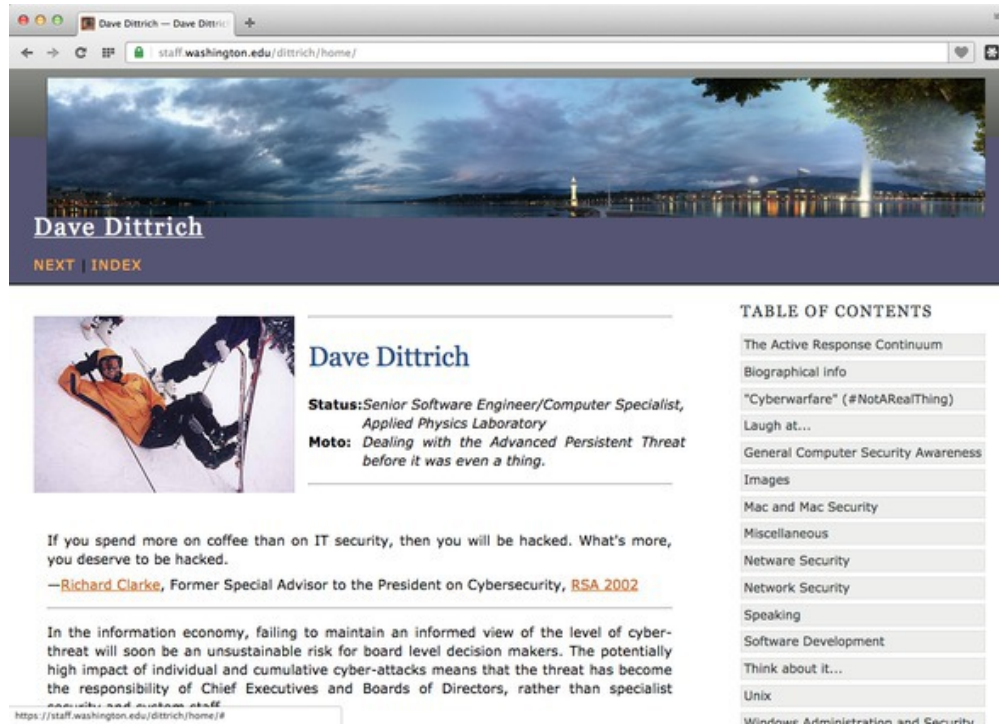


Fig. 5.3: A screen shot of Dave Dittrich's home page. Full size image

Note: The small version of an image can be created using ImageMagick's `convert` program like this:

```
$ convert DD_home_page.png -resize 50% DD_home_page_small.jpg
```

Section numbering

Sphinx does not render HTML documents with section numbers by default, but it will render LaTeX PDF documents with section numbers. To make these both consistent, add the `:numbered:` option to the `toctree` directive:

```
Contents:

.. toctree::
   :numbered:
   :maxdepth: 2
```

See <http://sphinx-doc.org/markup/toctree.html> and <http://stackoverflow.com/questions/20061577/sphinx-section-numbering-for-more-than-level-3-headings-sectnum>

Converting HTML content to Sphinx reST files

Many of the DIMS project documents are created using templates described in [A forgotten military standard that saves weeks of work \(by providing free project management templates\)](#), by Kristof Kovacs. Kovacs' web site has HTML versions of each of these templates in a ZIP archive. Download the archive and unpack it. Using the program `html2rest`, you can convert these documents to a reST format document.

Warning: The format of these HTML files is not parsed properly by `html2rest`, at least not without some pre-processing. Strip out the HTML break tags using the following commands:

```
$ sed 's|<br/>||g' ~/MIL-STD-498-templates-html-master/SRS.html > SRS.html
$ html2rest SRS.html > SRS.rst-orig
```

Once converted, you can now split the file `SRS.rst-orig` into separate sections, enable section numbering with the `:numbered:` option to the `toctree` directive and strip off the hard-coded numbers from sections, and add labels for sections (for cross-referencing). Here is an example of before and after for one such section from `SRS.html`:

Section 5 from the original HTML:

```
<h1>5. Requirements traceability.</h1>
<p>This paragraph shall contain:
<ol type="a">
  <li>Traceability from each CSCI requirement in this specification to the
  ↳system (or subsystem, if applicable) requirements it addresses. (Alternatively,
  ↳this traceability may be provided by annotating each requirement in Section 3.)

  Note: Each level of system refinement may result in requirements not directly
  ↳traceable to higher-level requirements. For example, a system architectural design
  ↳that creates multiple CSCIs may result in requirements about how the CSCIs will
  ↳interface, even though these interfaces are not covered in system requirements.
  ↳Such requirements may be traced to a general requirement such as "system
  ↳implementation" or to the system design decisions that resulted in their generation.
  ↳ </li>
  <li>Traceability from each system (or subsystem, if applicable) requirement
  ↳allocated to this CSCI to the CSCI requirements that address it. All system
  ↳(subsystem) requirements allocated to this CSCI shall be accounted for. Those that
  ↳trace to CSCI requirements contained in IRSs shall reference those IRSs.</li>
</ol>
</p>
```

Section 5 from `SRS.rst-orig` after conversion with `html2rest`:

```
5. Requirements traceability.
=====

This paragraph shall contain:

#. Traceability from each CSCI requirement in this specification to
the system (or subsystem, if applicable) requirements it addresses.
(Alternatively, this traceability may be provided by annotating each
requirement in Section 3.) Note: Each level of system refinement may
result in requirements not directly traceable to higher-level
requirements. For example, a system architectural design that creates
multiple CSCIs may result in requirements about how the CSCIs will
interface, even though these interfaces are not covered in system
requirements. Such requirements may be traced to a general requirement
such as "system implementation" or to the system design decisions that
resulted in their generation.
#. Traceability from each system (or subsystem, if applicable)
requirement allocated to this CSCI to the CSCI requirements that
address it. All system (subsystem) requirements allocated to this CSCI
shall be accounted for. Those that trace to CSCI requirements
contained in IRSs shall reference those IRSs.
```

Section 5 in a separate file `traceability.rst`:

```
.. _traceability:

Requirements traceability
=====

This paragraph shall contain:

#. Traceability from each CSCI requirement in this specification to
the system (or subsystem, if applicable) requirements it addresses.
(Alternatively, this traceability may be provided by annotating each
requirement in Section :ref:`requirements`.)

.. note::

    Each level of system refinement may result in requirements not directly
    traceable to higher-level requirements. For example, a system
    architectural design that creates multiple CSCIs may result in
    requirements about how the CSCIs will interface, even though these
    interfaces are not covered in system requirements. Such requirements may
    be traced to a general requirement such as "system implementation" or to
    the system design decisions that resulted in their generation.

#. Traceability from each system (or subsystem, if applicable)
requirement allocated to this CSCI to the CSCI requirements that
address it. All system (subsystem) requirements allocated to this CSCI
shall be accounted for. Those that trace to CSCI requirements
contained in IRSS shall reference those IRSS.
```

The rendered HTML for section 5 can be seen in the figure [Correct rendering of note within list](#).

Referencing subsections or figures

In the last example covered in section [Converting HTML content to Sphinx reST files](#), note the label definition `.. _traceability:` right before the section heading *Requirements traceability*. A reference to this label will result in the section heading being used as the text for the hyperlink. This section itself is preceded by the label `referencinglabels`, which is rendered on reference as [Referencing subsections or figures](#). This is the way to reference a sub-section (or figure, table, etc.) of a document.

Note: The section [Cross-referencing between documents with the `sphinx.ext.intersphinx` extension](#) builds on this concept of linking to arbitrary locations in a file by label.

Common Problems

Improperly referencing links to external documents

Sphinx documents are used to produce HTML, but reST itself is not like HTML in terms of links to external references. An HTML document may have many `HREF` elements that all have the same text to represent the hyperlink, but links in reST documents produce targets that can be cross-referenced from multiple places and need to be unique.

Here is output of `sphinx-autobuild` showing this problem:

```
+----- source/.vmprovisioning.rst.swp changed -----
/Users/dittrich/git/dims-ci-utils/docs/source/vmprovisioning.rst:3: WARNING:
↪Duplicate explicit target name: "here".
/Users/dittrich/git/dims-ci-utils/docs/source/vmprovisioning.rst:3: WARNING:
↪Duplicate explicit target name: "here".
```

This message reports that the target name *help* has been duplicated twice within the text (meaning it occurs three times in definitions).

Note: The line number reported is not accurate for some reason. It is not actually on line 3 in `vmprovisioning.rst`.

Here are the three occurrences of the target *help* in the file:

[...]

When a new VM is created from `base.ovf` via `import`, it will of course inherit the complete hard drive contents from the OVF. If the import is done without the ```keepnatmacs``` option, the new VM will have a new MAC address, which will then *not* match the details in the `udev` file, at which point the VM's network configuration will appear broken. This issue is a known one, see e.g. `here`

`<http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1032790>`_`
or simply Google **ubuntu udev vm**.

[...]

Again ```packer``` is used to build the ```base-keyed``` OVF from the base OVF. We also perform an extra step (see ```base-keyed/base-keyed.json```) in converting the OVF to a Vagrant box file. Further, we use the merged ```Vagrantfile``` idiom (Packer instructions `here`

`<https://www.packer.io/docs/post-processors/vagrant.html>`_` and Vagrant description `here`

`<https://docs.vagrantup.com/v2/vagrantfile/>`_`) to embed SSH credentials into the box file itself. These are then available to every Vagrant-managed VM created from that box file. This eliminates the need to manage (via manual or automated edits) SSH user name and private key name in each/every ```Vagrantfile``` spawned from this box.

[...]

These three links can all be made unique like this:

[...]

When a new VM is created from `base.ovf` via `import`, it will of course inherit the complete hard drive contents from the OVF. If the import is done without the ```keepnatmacs``` option, the new VM will have a new MAC address, which will then *not* match the details in the `udev` file, at which point the VM's network configuration will appear broken. This issue is a known one, as seen in the VMware Knowledge base article ```Networking fails after cloning an Ubuntu virtual machine (1032790)``` or simply Google **ubuntu udev vm**.

[...]

Again ``packer`` is used to build the ``base-keyed`` OVF from the base OVF. We also perform an extra step (see ``base-keyed/base-keyed.json``) in converting the OVF to a Vagrant box file. Further, we use the merged ``Vagrantfile`` idiom to embed SSH credentials into the box file itself. (See the Packer documentation on the ``Vagrant Post-Processor``_ and Vagrant documentation on the ``Vagrantfile``_.) These are then available to every Vagrant-managed VM created from that box file. This eliminates the need to manage (via manual or automated edits) SSH user name and private key name in each/every ``Vagrantfile`` spawned from this box.

[...]

```
.. _Networking fails after cloning an Ubuntu virtual machine (1032790): http://kb.
↪vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&
↪externalId=1032790
.. _Vagrant Post-Processor: https://www.packer.io/docs/post-processors/vagrant.html
.. _Vagrantfile: https://docs.vagrantup.com/v2/vagrantfile/
```

Not having the proper white space around literal blocks

We saw this example, and how to fix it, in Section *Fixing errors*.

Using inconsistent indentation in literal blocks and directives

Say you are trying to create a list, and you want to include notes or warnings in one of the list items. Here are examples of the wrong and right way to do this.

Source code (incorrect indentation within list):

```
.. _traceability:

Requirements traceability
=====

This paragraph shall contain:

#. Traceability from each CSCI requirement in this specification to
   the system (or subsystem, if applicable) requirements it addresses.
   (Alternatively, this traceability may be provided by annotating each
   requirement in Section :ref:`requirements`.)

.. note::

   Each level of system refinement may result in requirements not directly
   traceable to higher-level requirements. For example, a system
   architectural design that creates multiple CSCIs may result in
   requirements about how the CSCIs will interface, even though these
   interfaces are not covered in system requirements. Such requirements may
   be traced to a general requirement such as "system implementation" or to
   the system design decisions that resulted in their generation.

#. Traceability from each system (or subsystem, if applicable)
   requirement allocated to this CSCI to the CSCI requirements that
```

address it. All system (subsystem) requirements allocated to this CSCI shall be accounted for. Those that trace to CSCI requirements contained in IRSs shall reference those IRSs.

5. Requirements traceability

This paragraph shall contain:

1. Traceability from each CSCI requirement in this specification to the system (or subsystem, if applicable) requirements it addresses. (Alternatively, this traceability may be provided by annotating each requirement in Section [Requirements](#).)

Note: Each level of system refinement may result in requirements not directly traceable to higher-level requirements. For example, a system architectural design that creates multiple CSCIs may result in requirements about how the CSCIs will interface, even though these interfaces are not covered in system requirements. Such requirements may be traced to a general requirement such as "system implementation" or to the system design decisions that resulted in their generation.

1. Traceability from each system (or subsystem, if applicable) requirement allocated to this CSCI to the CSCI requirements that address it. All system (subsystem) requirements allocated to this CSCI shall be accounted for. Those that trace to CSCI requirements contained in IRSs shall reference those IRSs.

Fig. 5.4: Incorrect rendering of note within list

Source code (correct indentation within list):

```
.. _traceability:

Requirements traceability
=====

This paragraph shall contain:

#. Traceability from each CSCI requirement in this specification to
the system (or subsystem, if applicable) requirements it addresses.
(Alternatively, this traceability may be provided by annotating each
requirement in Section :ref:`requirements`.)

.. note::

    Each level of system refinement may result in requirements not directly
    traceable to higher-level requirements. For example, a system
    architectural design that creates multiple CSCIs may result in
    requirements about how the CSCIs will interface, even though these
    interfaces are not covered in system requirements. Such requirements may
    be traced to a general requirement such as "system implementation" or to
    the system design decisions that resulted in their generation.

#. Traceability from each system (or subsystem, if applicable)
requirement allocated to this CSCI to the CSCI requirements that
address it. All system (subsystem) requirements allocated to this CSCI
shall be accounted for. Those that trace to CSCI requirements
contained in IRSs shall reference those IRSs.
```

Having multiple colons in link target labels

It is easy to get used to using directives like `.. figure::` or `.. note::` and placing the double-colon after them. Labels look similar, but are not directives. They also have the underscore in front of them and should look like:

5. Requirements traceability ¶

This paragraph shall contain:

1. Traceability from each CSCI requirement in this specification to the system (or subsystem, if applicable) requirements it addresses. (Alternatively, this traceability may be provided by annotating each requirement in [Section Requirements](#).)

Note: Each level of system refinement may result in requirements not directly traceable to higher-level requirements. For example, a system architectural design that creates multiple CSCIs may result in requirements about how the CSCIs will interface, even though these interfaces are not covered in system requirements. Such requirements may be traced to a general requirement such as "system implementation" or to the system design decisions that resulted in their generation.

2. Traceability from each system (or subsystem, if applicable) requirement allocated to this CSCI to the CSCI requirements that address it. All system (subsystem) requirements allocated to this CSCI shall be accounted for. Those that trace to CSCI requirements contained in IRSs shall reference those IRSs.

Fig. 5.5: Correct rendering of note within list

```
.. _label:

This is a reference to :ref:`label`.
```

Advanced Use of Sphinx Features

This section discusses more advanced features of Sphinx to accomplish particular tasks.

To illustrate two cases, consider the following:

- During DIMS development, there will be (1) a DIMS instance for developers to use, (2) another DIMS instance for test and evaluation prior to release, and (3) yet another instance for user acceptance and functional testing that will be used by the PRISEM user base.
- In production, there will be an instance of DIMS deployed for different groups in multiple parts of the country, each with their own name, organizational structure and policies, etc.

In order to produce documentation that provides a sufficient level of precise detail so as to be immediately useful, documents for DIMS will need to be build by doing parameterized construction of documentation based on case-specific parameters.

Put yourself in a DIMS user's shoes. Which of the following two examples would be more useful to you?

Note: To access the DIMS front end, connect your browser to: `https://dims.example.com:12345/dimsapi/`

Note: To access the DIMS front end, connect your browser to the specific host and port configured for the login portal server, followed by the string `"/dimsapi/"`. Ask your site administrator for the details.

Every instantiation of the full DIMS system (comprised of many separate service components) will be unique in several run-time aspects. Each will have its own IP address block, its own top level Domain Name System name, its own organizational name, its own policies and its own membership. That is just a start. One set of documentation cannot possibly be generalized in a way that it can be used by everyone, without reading like the second example above. Each instantiation needs its own uniquely produced documentation, which means **documentation must be**

configured just like the system itself is configured. If the documentation must be hand-edited for each user, that places a huge burden on those wanting to implement DIMS and the system will not be used widely enough to have the intended impact.

Cross-referencing between documents with the `sphinx.ext.intersphinx` extension

ReST supports [Cross-referencing arbitrary locations](#) within a document using `:ref:`. To reference arbitrary locations (by their label) in other documents requires the Sphinx extension `sphinx.ext.intersphinx`. (See the documentation for `sphinx.ext.intersphinx` and Section [Referencing subsections or figures](#) for more on labels.)

Intersphinx links allow, for example, cross referencing a test in the Test Plan document to a requirement or user story in the Requirements document to provide requirements traceability in testing.

Mapping URLs to documents

The first step is to enable the extension by making sure it is included in the `conf.py` file:

```
# Add any Sphinx extension module names here, as strings. They can be
# extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
# ones.
extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.doctest',
    'sphinx.ext.todo',
    'sphinx.ext.intersphinx',
    'sphinx.ext.graphviz',
    'sphinx.ext.ifconfig',
]
```

When you build HTML output, any labels that are defined in your reST files are recorded in an *inventory* file. By default, the inventory is named `objects.inv`.

To cross-reference the `objects.inv` files from other documents requires a mapping of these inventories to symbolic name to define a label namespace for use in `:ref:` directives.

Note: You may use multiple targets for each inventory file, which is necessary when you are building multiple documents locally before they have been published in their final internet-accessible web site (e.g, the Read the Docs site). Obviously, if the remote inventory does not exist, it cannot be used (but when it does exist, you may want to use it instead of a local copy). Documents built automatically for publication with Jenkins would not have local copies, so they automatically would link with remote versions.

Warning: Because of the *chicken/egg* problem just described, document sets that are to be cross-linked would need to be rendered *twice* in order to first generate the inventory file that is used by other documents that reference it, and to get those inventories from the other documents in order to reference them. This is similar to how LaTeX works, where the recommendation is to run `pdflatex` twice, then run `bibtex` for bibliographies, then run `pdflatex` one last time to get cross-references and citations set up properly.

In the example below, both *local* and *remote* locations are specified.

Warning: You cannot use '-' in the symbol that maps the inventory files, so the following examples simply remove that character from the Git repo names.

```
intersphinx_cache_limit = -1    # days to keep the cached inventories (0 == forever)
intersphinx_mapping = {
    'dimsocd': ('http://u12-dev-svr-1.prisem.washington.edu:8080/docs/develop/
↪html/dims-ocd',
                ('../../dims-ocd/build/html/objects.inv', None)),
    'dimsad': ('http://u12-dev-svr-1.prisem.washington.edu:8080/docs/develop/html/
↪dims-ad',
               ('../../dims-ad/build/html/objects.inv', None)),
    'dimstp': ('http://u12-dev-svr-1.prisem.washington.edu:8080/docs/develop/html/
↪dims-tp',
               ('../../dims-tp/build/html/objects.inv', None))
}
```

Linking to the label

In the reST document (in this case, the `referenceddocs.rst` file), normal `:ref:` directives are used, but the target of the `:ref:` includes the name of the inventory prepended to the label so as to map to the proper URL. The first reference in this example maps to the Operational Concept Description document:

```
.. _referenceddocs:

Referenced Documents
=====

The following documents describe the DIMS project and provide background
material related to tasking.

#. :ref:`dimsocd:dimsoperationalconceptdescription`

#. :ref:`dimsad:dimsarchitecturedesign`

#. :ref:`dimstp:dimstestplan`

#. HSHQDC-13-C-B0013, "From Local to Gobal Awareness: A Distributed Incident_
↪Management System," Draft contract, Section C - Statement of Work (marked up_
↪version)

#. MIL-STD-498, Military Standard Software Development and Documentation,
   AMSC No. N7069, Dec. 1994.
```

The label `dimsoperationalconceptdescription` occurs in the `index.rst` file on line 3, immediately preceding the title on line 6 (which has the release number inserted into it).

```
1  .. DIMS Operational Concept Description documentation master file.
2
3  .. _dimsoperationalconceptdescription:
4
5  =====
6  DIMS Operational Concept Description v |release|
7  =====
8
9  .. topic:: Executive Summary
```


Since HSPD-7 was released in 2003, the Department of Homeland Security has had a core mission of working to protect the nation's critical infrastructure. In 2008, the *National Response Framework* was released, and ...

The final rendered *DIMS System Requirements* document has links to the related *DIMS Operational Concept Description*, *DIMS Architecture Design*, and *DIMS Test Plan* documents, all with their current release number visible for precise cross-referencing.

Note: Documents released from the master branch, all at once, will be easier to trace back to the code base for which they apply.

The screenshot shows a web interface for 'DIMS System Requirements'. On the left is a sidebar with a search bar and a list of sections: 1. Scope, 2. Referenced Documents (highlighted), 3. Requirements, 4. Qualification provisions, 5. Notes, 6. License, and 7. Appendices. The main content area is titled '2. Referenced Documents' and includes a 'View page source' link. Below the title, it states: 'The following documents describe the DIMS project and provide background material related to tasking.' A list of five references follows, each with a link to its respective document: 1. DIMS Operational Concept Description v 2.1.9, 2. DIMS Architecture Design v 2.3.9, 3. DIMS Test Plan v 2.0.23, 4. HSHQDC-13-C-B0013, "From Local to Global Awareness: A Distributed Incident Management System," Draft contract, Section C - Statement of Work (marked up version), and 5. MIL-STD-498, Military Standard Software Development and Documentation, AMSC No. N7069, Dec. 1994.

Fig. 5.6: Rendered intersphinx links

When you build the document, you will see the `objects.inv` files being loaded:

```
(dimsenv)[dittrich@localhost dims-sr (develop)]$ make html
Makefile:27: warning: overriding commands for target `help'
/opt/dims/etc/Makefile.dims.global:48: warning: ignoring old commands for target `help'
↪
sphinx-build -b html -d build/doctrees    source build/html
Running Sphinx v1.3.1+
loading pickled environment... done
loading intersphinx inventory from http://u12-dev-svr-1.prisem.washington.edu:8080/
↪docs/develop/html/dims-ocd/objects.inv...
loading intersphinx inventory from http://u12-dev-svr-1.prisem.washington.edu:8080/
↪docs/develop/html/dims-ad/objects.inv...
loading intersphinx inventory from http://u12-dev-svr-1.prisem.washington.edu:8080/
↪docs/develop/html/dims-tp/objects.inv...
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 0 source files that are out of date
updating environment: [config changed] 8 added, 0 changed, 0 removed
reading sources... [ 12%] appendices
reading sources... [ 25%] index
...
```

Insertion of text using direct substitution

Sphinx has ways of producing customized output when documents are built using direct textual substitution, and through execution of programs from within Sphinx. The simplest method is direct substitution.

Say you want a copyright symbol in a document. You start by selecting or creating a file that maps strings surrounded by pipe characters to some other string. There is a file called `isonum.txt` that does this for many Unicode characters, like the copyright symbol. The first 20 lines of this file look like this:

```

1  .. This data file has been placed in the public domain.
2  .. Derived from the Unicode character mappings available from
3     <http://www.w3.org/2003/entities/xml/>.
4     Processed by unicode2rstsubs.py, part of Docutils:
5     <http://docutils.sourceforge.net>.
6
7  .. |amp|    unicode:: U+00026 .. AMPERSAND
8  .. |apos|   unicode:: U+00027 .. APOSTROPHE
9  .. |ast|    unicode:: U+0002A .. ASTERISK
10 .. |brvbar|  unicode:: U+000A6 .. BROKEN BAR
11 .. |bsol|   unicode:: U+0005C .. REVERSE SOLIDUS
12 .. |cent|   unicode:: U+000A2 .. CENT SIGN
13 .. |colon|  unicode:: U+0003A .. COLON
14 .. |comma|  unicode:: U+0002C .. COMMA
15 .. |commat| unicode:: U+00040 .. COMMERCIAL AT
16 .. |copy|   unicode:: U+000A9 .. COPYRIGHT SIGN
17 .. |curren| unicode:: U+000A4 .. CURRENCY SIGN
18 .. |darr|   unicode:: U+02193 .. DOWNWARDS ARROW
19 .. |deg|    unicode:: U+000B0 .. DEGREE SIGN
20 .. |divide| unicode:: U+000F7 .. DIVISION SIGN

```

Note: This is how to visually parse line 16: The `..` at the start to indicate a reST directive is being used, `|copy|` as the string to match, `unicode:: U+000A9` as a reST directive for a Unicode character, and `.. COPYRIGHT SIGN` as a comment that explains this is the copyright sign. The comment is unnecessary, but helps explain what is being mapped.

You must first include the map before any substitutions will be recognized, then wherever the string `|copy|` occurs, the Unicode character `U+000A9` will be inserted. Here is a simple example of how to do this:

```

.. include:: <isonum.txt>

Copyright |copy| 2015 University of Washington. All rights reserved.

```

This code renders as follows:

Copyright © 2015 University of Washington. All rights reserved.

Insertion of text programmatically

A more complicated way of text substitution is by using the fact that Sphinx is a Python program, which can include and execute Python code at run time.

Let's start by creating a minimal Sphinx doc set using `sphinx-quickstart`.

We then modify the `conf.py` file by uncommenting the path modification line as follows:

```
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown here.
sys.path.insert(0, '/opt/dims/etc')
```

Next, put this line at the very end of the file:

```
from rst_prolog import *
```

Create the file `/opt/dims/etc/rst_prolog.py` and insert an `rst_prolog` string that is used by Sphinx before generating any output files:

```
rst_prolog = """
.. |dims_ftw|      replace:: for the win
"""
```

Here is a minimal Sphinx file that includes the variable that we will substitute at compile-time:

```
.. Sphinx Demo repository documentation master file, created by
   sphinx-quickstart on Tue Dec 30 12:43:11 2014.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to Sphinx Demo repository's documentation!
=====

.. toctree::
   :maxdepth: 2

.. include: <rst_prolog>

This is |dims_ftw|!!!

Indices and tables
=====

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

When you render this file with `make html` and then load it with a browser (in this case, `lynx`), you get the following:

```
#Welcome to Sphinx Demo repository's documentation! -- Sphinx.. (p1 of 2)
#Sphinx Demo Repository 1.0 documentation

Navigation

    * index
    * Sphinx Demo Repository 1.0 documentation

Welcome to Sphinx Demo repository's documentation!

This is for the win!!!

Indices and tables

    * Index
```

- * Module Index
- * Search Page

Table Of Contents

-- press space for next page --

Arrow keys: Up and Down to move. Right to follow a link; Left to go back.

H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list

Warning: When you render this document, Python includes the `rst_prolog.py` file (which is actually Python code) and will produce a `.pyc` file. You may need to delete it, if and when you remove the associated `.py` file.

```
[dittrich@localhost docs (dev)]$ ls -l /opt/dims/etc/
total 24
-rw-r--r--  1 dims      dims  3046 Dec 30 10:11 Makefile.dims.global
-rw-r--r--  1 dittrich  dims   58 Dec 30 15:13 rst_prolog.py
-rw-r--r--  1 dittrich  dims  177 Dec 30 15:14 rst_prolog.pyc
```

Inserting a graph using Graphviz

Sphinx uses [Graphviz](#) to render directed and undirected graphs inline in a document. To insert a graph, create a [DOT](#) language file to describe the graph, then reference the file using the `graphviz::` directive.

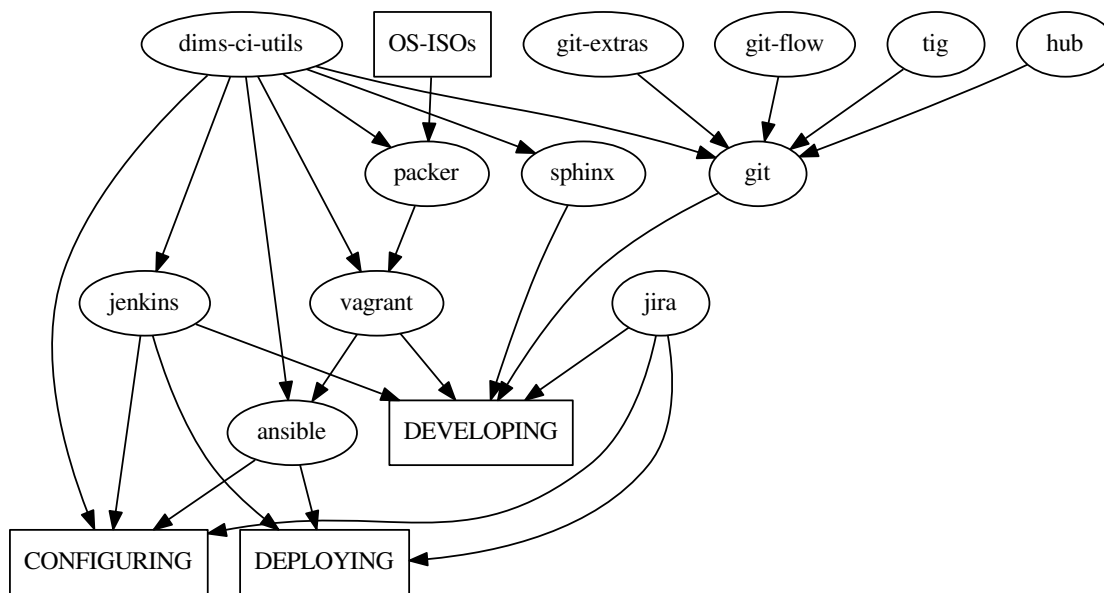


Fig. 5.7: Relationships between tools and processes in DIMS

The [DOT](#) file for the graph above looks like this:

```
digraph tools {
    "DEVELOPING" [shape=rectangle];
```

```
"DEVELOPING" [shape=rectangle];
"CONFIGURING" [shape=rectangle];
"OS-ISOs" [shape=rectangle];
"dims-ci-utils" -> "jenkins";
"dims-ci-utils" -> "ansible";
"dims-ci-utils" -> "packer";
"dims-ci-utils" -> "vagrant";
"dims-ci-utils" -> "sphinx";
"dims-ci-utils" -> "git";
"git-extras" -> "git";
"git-flow" -> "git";
"tig" -> "git";
"hub" -> "git";
"OS-ISOs" -> "packer";
"packer" -> "vagrant";
"vagrant" -> "ansible";
"vagrant" -> "DEVELOPING";
"git" -> "DEVELOPING";
"sphinx" -> "DEVELOPING";
"jira" -> "DEVELOPING";
"jira" -> "DEPLOYING";
"jira" -> "CONFIGURING";
"jenkins" -> "DEVELOPING";
"ansible" -> "DEPLOYING";
"jenkins" -> "DEPLOYING";
"dims-ci-utils" -> "CONFIGURING";
"ansible" -> "CONFIGURING";
"jenkins" -> "CONFIGURING";
}
```

Note: You can find a [Gallery](#) of example DOT files at the [Graphviz web site](#) that shows how to do more advanced things, such as labelled edges.

Continuous Integration

Continuous Integration

Continuous Integration is a software engineering process where multiple developers merge their working code into a coherent system on a regular basis, allowing for easier testing of code changes and integration of disparate parts of the system. Using a combination of a build system (Jenkins, in this case) and triggers invoked by the source code management system (Git, in this case), a change to source code results in that code being compiled, bundled, and installed (as necessary) onto the hosts where it needs to run to serve its function within the system as a whole.

Continuous Integration works well in a software engineering environment using [Agile/Scrum](#).

How source changes are propagated

This section summarizes how changes in source repos are propagated using Jenkins and Ansible. You can find more information in the documentation for the `ansible-inventory` and `ansible-playbooks` repositories.

Git repos containing DIMS software under development contain “post-receive” hooks which notify the Jenkins server when changes are pushed to a repository. We are currently using two kinds of hooks: 1) A general hook which notifies Jenkins that a push has occurred, and 2) A hook which calls a parameterized Jenkins job when a push has occurred.

For the general hook, Jenkins jobs essentially “listen” for the notifications. A Jenkins job specifies the repository and branch it wishes to be notified about, as well as optionally specifying particular directory locations it is monitoring. When a notification is received that matches, the job will determine if any actually source changes occurred. If so, the job is run.

The “parameterized” hook is used to call a parameterized Jenkins documentation job when a push is received in a system documentation repository. The Jenkins job builds the documentation in the repo and deploys it (using Ansible) to any documentation servers in the system inventory that correspond to the branch that was updated.

Attention: In general, each repository with DIMS software under development will have a Jenkins job “listening” for each branch of the repository that we want to build and deploy continuously. Note that Jenkins jobs can be

triggered by changes to more than one branch, but we found it is unreliable. When using hubflow to do releases, for example, a job that was supposed to be triggered by changes in both the `master` and `develop` branch only built the `develop` branch even though changes had been pushed to both the `master` and `develop` branches. Since we can programmatically create jobs via the Jenkins Job DSL plugin, it is trivial to create (and modify) jobs for both the `master` and `develop` branches (and other branches as needed - release branches for testing, for example).

A Jenkins job that builds and deploys updated software from a Git repository uses Ansible to do the deployment.

Note: We are currently using flat files to define the inventory for a deployment (a “host” file), although we hope to move to using dynamic inventories. Either way, we need to define the hosts in a system and group them in ways that make deployments easy and scalable. (More on this subject can be found in the `ansibleinventory:ansibleinventory` documentation.)

Ideally, software in a “develop” branch would be deployed in one or more development and/or test systems, each defined by a single host file (inventory). Software in the “master” branch would be deployed to one or more production or operational systems. One could set up a workflow where release branches were automatically deployed to a release-test system - where the software could be tested before final release. (When the code in the release branch was fully tested and accepted, it would be merged into master according to the hubflow workflow, which would cause it to be automatically deployed to production/operational systems).

Figure *How software in repositories flows to machines in inventories* illustrates this. At the current time, however, we essentially only have one “system” - a “development” system that has grown ad hoc and was not created from scratch using our proposed workflows. The figure shows how we have develop branches of (some) repos also installed in what we’ve named “prisen”, “project”, and “infrastructure” inventories. Ideally we would want to consolidate machines under the “development” inventory if we truly wish to install “develop” branch software automatically on all these machines. This would make defining jobs in the Jenkins DSL simpler as well. See the `ansibleinventory:ansibleinventory` documentation for a description of our current inventory.

We define “groups” for machines in inventories. The groups are used by Ansible to determine whether or not plays should be run on machines in an inventory. The following figure illustrates this. Machines can be found in more than one group. The group “all” contains all machines in the inventory. A playbook that specifies a host of “all” will run on all machines in the inventory (unless further limited by other means, such as flags passed to the `ansible-playbook` command or conditional expressions in a role task). The `dims-ci-utils` code, for example, is to be installed on all machines in the inventory. However, the role that deploys `dims-ci-utils` restricts a couple tasks to specific groups of machines. One of those groups is the “git” group.

Continuous deployment of documentation

For our documentation, we currently deploy all docs from all repository branches to a single VM to make retrieval efficient and to aid in development of the documentation. Ansible is not used for deployment. We simply use `rsync` over SSH to deploy the docs.

The following figure shows the flows involved in documentation deployment.

The workflow runs something like this:

1. Push to remote repository runs a post-receive hook.
2. Post-receive hook calls the parameterized Jenkins job `dims-docs-deploy` if either a branch is deleted or if a branch is updated in a repo that contains documentation. The job is called twice - once to build html and once to build PDF.

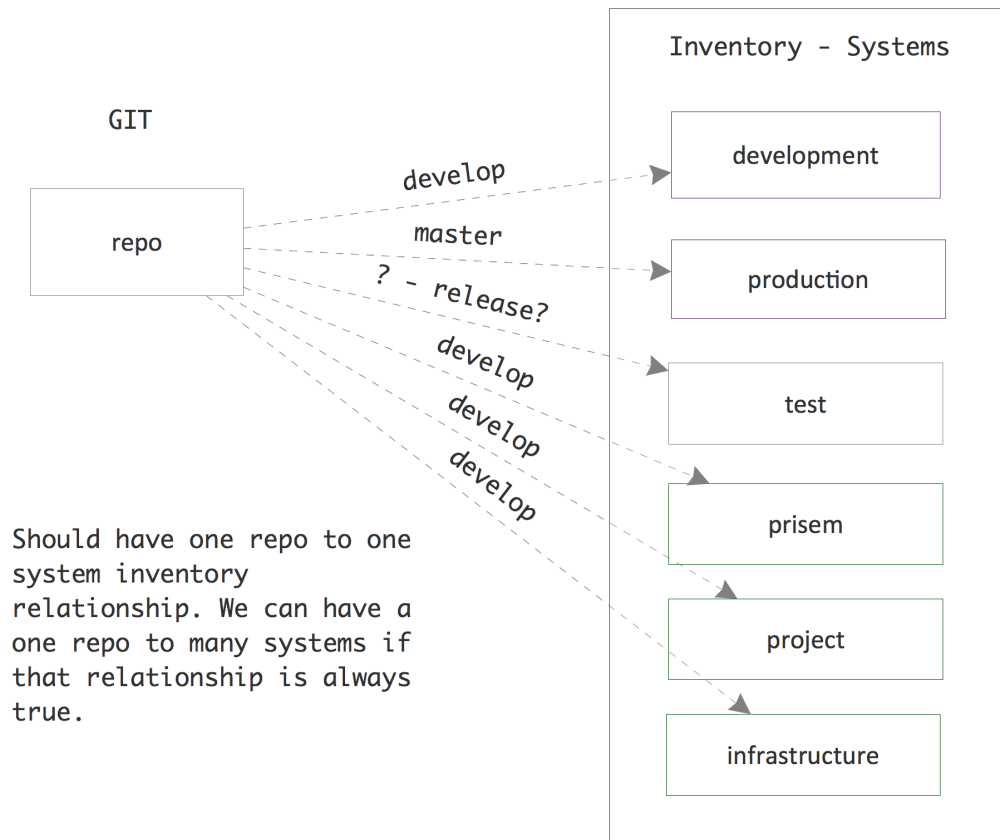


Fig. 6.1: How software in repositories flows to machines in inventories

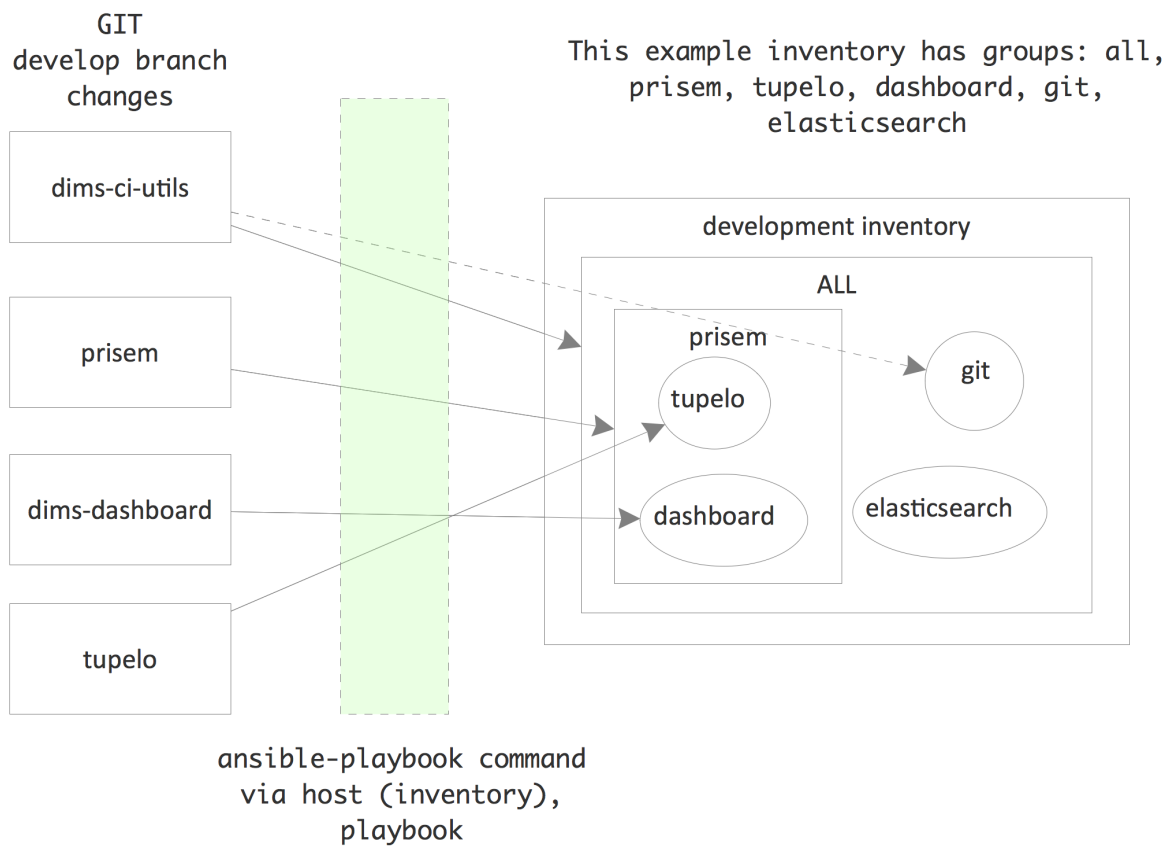


Fig. 6.2: Machines belong to different groups in an inventory

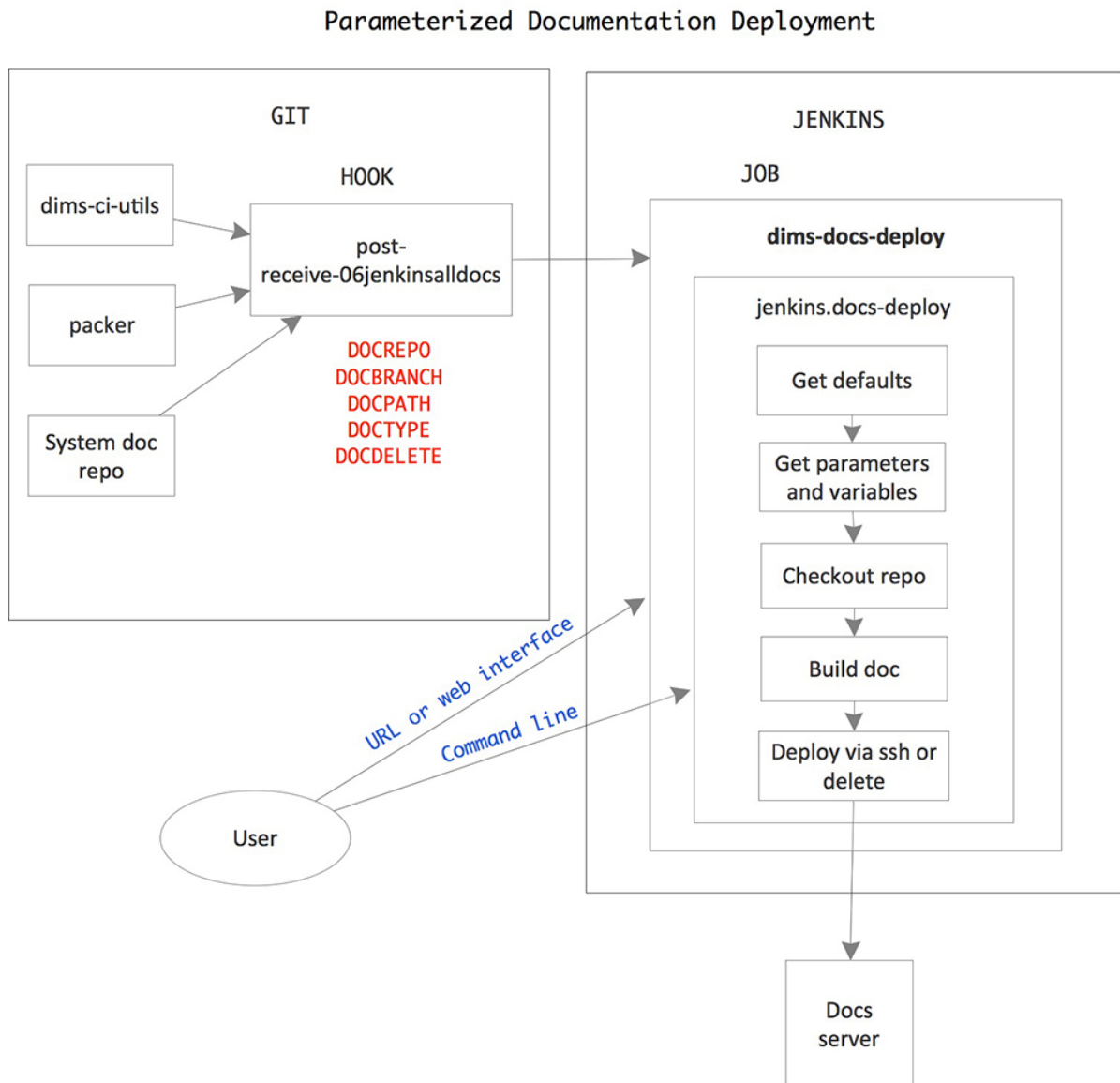


Fig. 6.3: Diagram of documentation deployment flow.

3. Jenkins job `dims-deploy-docs` runs the script `jenkins.docs-deploy`
4. Script `jenkins.docs-deploy` clones and checks out the documentation, builds the documentation, and rsyncs the documentation to the target server.

Documentation is deployed on the target documentation server with the following directory structure:

```
/opt/dims/docs/$BRANCH/html/$REPONAME  
/opt/dims/docs/$BRANCH/pdf/$REPONAME
```

Note: `$BRANCH` only includes the last part of a branch name with the `/` delimiter. Therefore, since we use the hubflow branching model, branch `feature/dims-313` is deployed to `/opt/dims/docs/dims-313/html/$REPONAME` and `/opt/dims/docs/dims-313/pdf/$REPONAME`

To view the documentation, you go to `https://\protect{T1}\textdollarHOST:\protect{T1}\textdollarPORT/docs/\protect{T1}\textdollarBRANCH/\protect{T1}\textdollarTYPE/\protect{T1}\textdollarREPONAME` or go to `https://\protect{T1}\textdollarHOST:\protect{T1}\textdollarPORT/docs/` and browse the directory tree.

Currently the Jenkins job defaults to deploying the documentation on `https://u12-dev-svr-1.prisem.washington.edu:8443/docs`

The following paragraphs describe this workflow in more detail.

Post-receive hook

The post-receive hook, `post-receive-jenkins06alldocs`, calls a parameterized Jenkins job, `dims-docs-deploy`, when the repository receives a push. The hook code follows:

The hook determines if the repo contains documentation based upon the existence of the file `$REPO/docs/source/conf.py`. This determines the value of `DOCPATH`, which is the path in the repository to the Makefile that will build the docs.

Attention: All DIMS source repositories *must* have their documentation in a subdirectory named `docs/` in order to simplify the logic of finding and processing Sphinx documentation.

Once the `DOCPATH` is determined, two `curl` commands are sent to Jenkins server to call the job `dims-docs-deploy` - once for HTML, and once for PDF.

The hook source is located in `dims-ci-utils/git/` and is deployed by the `dims-ci-utils-deploy-$BRANCH` jobs.

Note: Currently, we do not have an automated way to add the symlink to the appropriate repos. The command to do that is:

```
$ ln -s /opt/git/bin/post-receive-06jenkinsalldocs /opt/git/${REPO}.git/hooks/post-  
receive-06jenkinsalldocs
```

Jenkins parameterized job `dims-docs-deploy`

The parameterized job `dims-docs-deploy` accepts the following parameters, with the defaults shown. All parameters are string parameters.

NAME	Default Value	Description
REPO		Repository to build
BRANCH		Branch to build
DOCPATH	.	Path to Makefile
DOCTYPE	html	Type of doc to build, html or pdf
DOCDELETE	false	True to delete docs for this branch
DOCHOST	u12-dev-svr-1.prisem.washington.edu	Host to receive the docs
DOCDEST	/opt/dims/docs	Root path on host to receive the docs
DOCURL	http://u12-dev-svr-1.prisem.washington.edu:8443/docs	URL of docs index

Defaults are given to make it easier to run the job via curl or via the Jenkins web interface - you don't need to include all of the parameters unless they are different than the defaults shown. The post-receive hooks sends the parameters REPO, BRANCH, DOCPATH, DOCTYPE, and DOCDELETE when it calls the job.

The `dims-docs-deploy` job is created via Jenkins DSL, so it is easy to modify if needed. The Jenkins DSL is located in the file `jenkins/DSL/jenkins-dsl.groovy`, in the `dims-ci-utils` repo. It is automatically run by the Jenkins seed job `dims-seed-job` whenever a change is pushed to the `jenkins/DSL` directory. In this way, the jobs are always up-to-date.

The portion of `jenkins-dsl.groovy` that builds the parameterized documentation job is shown below:

```
// Parameterized job to build and deploy DIMS documentation
job {
    name 'dims-docs-deploy'
    description ('Job to build and deploy DIMS documenation')
    logRotator(-1, 15, -1, -5)
    parameters {
        stringParam('REPO', '', 'Repository to build')
        stringParam('BRANCH', '', 'Branch of the repo to use')
        stringParam('DOCPATH', '.', 'Path to the doc Makefile from repo root')
        stringParam('DOCTYPE', 'html', 'Type of document to build - html or pdf')
        stringParam('DOCDELETE', 'false', 'True if the documentation is to be deleted
→ ')
        stringParam('DOCHOST', docHost, 'Host to receive the docs')
        stringParam('DOCDEST', docDest, 'Root destination on host to deploy the docs')
        stringParam('DOCURL', docUrl, 'URL to documentation root directory')
    }
    wrappers {
        preBuildCleanup()
    }
    // This job runs a script
    steps {
        shell ( "jenkins.dims-docs-deploy" )
    }
    publishers {
        downstreamParameterized postNotify
    }
}
```

The post-receive hook calls `dims-deploy-docs` via curl. You can also do this manually. For example:

```
$ curl --data-urlencode "REPO=${REPONAME}" --data-urlencode "BRANCH=${BRANCH}" --data-
→urlencode "DOCPATH=${DOCPATH}" --data-urlencode "DOCTYPE=${DOCTYPE}" $JENKINSURL/
→job/$JOB/buildWithParameters
```

where you have defined the variables shown and `JOB="dims-docs-deploy"` and `JENKINSURL="http://jenkins.prisem.washington.edu"`

You can also run the job via the Jenkins UI. Go to <http://jenkins.prisem.washington.edu/view/Current/job/>

`dims-docs-deploy/` and click the `Build with Parameters` link on the left.

Deployment script `jenkins.dims-docs-deploy`

As you can see in the previous section, the build step of the `dims-docs-deploy` job calls the `jenkins.dims-docs-deploy` script. The script has access to the job's parameters as environment variables, so they don't need to be passed explicitly when the script is called from the Jenkins job. The script, `jenkins.dims-docs-deploy`, along with other scripts used to build and deploy software by Jenkins, has its source located in `dims-ci-utils/jenkins/job-scripts`. It is deployed on Jenkins in the `/opt/dims/bin` directory.

The `jenkins.dims-docs-deploy` script follows the pattern used by other deploy job scripts:

1. Get default variables
2. Get parameters and message as needed
3. Checkout the docs repo and branch as specified by parameters
4. Build the docs
5. Deploy the docs

Since we are deploying all documentation to one server irrespective of branch, we do not use the Ansible infrastructure for final deployment. Instead we simply use `ssh` to make the modifications on the target machine as necessary. A variable, `REMOTEUSER`, is used for the user making the SSH calls. On Jenkins, this user is `ansible`. If you are running the script manually (while testing, for example), you can provide a different user by calling the script with `REMOTEUSER`, as in:

```
$ REPO=dims-sr BRANCH=develop DOCPATH=. DOCTYPE=html REMOTEUSER=$USER jenkins.dims-  
→docs-deploy
```

Of course, `$USER` must be a DIMS user on the target machine (one of the default users installed by Ansible when a DIMS machine is provisioned) and have the appropriate private key.

For your reference, the `jenkins.dims-docs-deploy` source follows:

Developing modules for the DIMS CLI app (dimscli)

Bootstrapping the dimscli app for development

1. Clone the repo `python-dimscli` from `git.prisem.washington.edu`. This can be done by running `dims.git.syncrepos`:
2. Prepare a new Python virtual environment with all of the DIMS pre-requisite tools necessary for DIMS software development:

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ VENV=dimscli_
↪dimsenv.install.user
sudo password:

PLAY [Install python virtual environment] *****

...

PLAY RECAP *****
localhost                : ok=30   changed=19   unreachable=0   failed=0
```

The new `dimscli` virtual environment should show up as an option for `workon`:

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ workon
dimscli
dimsenv
```

3. Invoke the new `dimscli` Python virtual environment.

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ workon dimscli
[+++] Virtual environment 'dimscli' activated [ansible-playbooks v1.2.113]
```

4. Because this is a new Python virtual environment created with the DIMS build tools, it only has those Python packages defined in Ansible playbooks role `python-virtualenv`.

The first time you try to run `dimscli`, or any time that you change any of the pre-requisites used for programming `dimscli` modules, you must use `pip` to update and/or install the required packages. These will eventually be added to the defaults for the `dimsenv` standard virtual environment.

```
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ pip install -U
-r requirements.txt
Collecting pbr<2.0,>=1.4 (from -r requirements.txt (line 1))
Using cached pbr-1.8.1-py2.py3-none-any.whl
Collecting six>=1.9.0 (from -r requirements.txt (line 2))
Using cached six-1.10.0-py2.py3-none-any.whl
Requirement already up-to-date: Babel>=1.3 in /home/dittrich/dims/envs/dimscli/
lib/python2.7/site-packages (from -r requirements.txt (line 3))
Collecting cliff>=1.14.0 (from -r requirements.txt (line 4))
Downloading cliff-1.15.0-py2-none-any.whl
Collecting keystoneauth1>=1.0.0 (from -r requirements.txt (line 5))
Downloading keystoneauth1-1.2.0-py2.py3-none-any.whl (149kB)
100% | extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock| 151kB 2.7MB/s
Collecting os-client-config!=1.6.2,>=1.4.0 (from -r requirements.txt (line 6))
Downloading os_client_config-1.10.1-py2.py3-none-any.whl (42kB)
100% | extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock extblock extblock
extblock extblock extblock extblock extblock extblock| 45kB 6.0MB/s
...

Running setup.py bdist_wheel for msgpack-python
Stored in directory: /home/dittrich/.cache/pip/wheels/f3/97/a5/
dd6e3b680de10b689464c44bc211239d1fe54bd296ff860897
Running setup.py bdist_wheel for functools32
Stored in directory: /home/dittrich/.cache/pip/wheels/38/c6/c7/
ee17acd621120c302e25c2fa8b3a8b235d5d1137c6ab4c9728
Successfully built simplejson warlock msgpack-python functools32
Installing collected packages: msgpack-python, oslo.serialization, python-
keystoneclient, simplejson,
python-neutronclient, functools32, jsonschema, jsonpointer, jsonpatch, warlock,
python-glanceclient,
python-novaclient, python-cinderclient, python-openstackclient

Successfully installed functools32-3.2.3.post2 jsonpatch-1.12 jsonpointer-1.10
jsonschema-2.5.1 msgpack-python-0.4.6
oslo.serialization-1.11.0 python-cinderclient-1.4.0 python-glanceclient-1.1.0
python-keystoneclient-1.8.1
python-neutronclient-3.1.0 python-novaclient-2.34.0 python-openstackclient-1.8.0
simplejson-3.8.1 warlock-1.2.0
PrettyTable-0.7.2 appdirs-1.4.0 cliff-1.15.0 cliff-tablib-1.1 cmd2-0.6.8
debtcollector-0.10.0 iso8601-0.1.11
keystoneauth1-1.2.0 monotonic-0.4 netaddr-0.7.18 netifaces-0.10.4 os-client-
config-1.10.1 oslo.config-2.6.0
oslo.i18n-2.7.0 oslo.utils-2.7.0 oslosphinx-3.3.1 pbr-1.8.1 pyparsing-2.0.5 pytz-
2015.7 requests-2.8.1
six-1.10.0 stevedore-1.9.0 tablib-0.10.0 unicodecsv-0.14.1 wrapt-1.10.5
```

5. Once all the pre-requisite packages are installed in the virtual environment, install the `dimscli` app and its modules as well using `python setup.py install` or `pip install -e .` (either will work):


```
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ python setup.
↳py install
running install
[pbr] Writing ChangeLog
[pbr] Generating ChangeLog
[pbr] ChangeLog complete (0.0s)
[pbr] Generating AUTHORS
[pbr] AUTHORS complete (0.0s)
running build
running build_py
creating build
creating build/lib
creating build/lib/dimscli
creating build/lib/dimscli/common

...

byte-compiling /home/dittrich/dims/envs/dimscli/lib/python2.7/site-packages/
↳dimscli/common/timing.py to timing.pyc
byte-compiling /home/dittrich/dims/envs/dimscli/lib/python2.7/site-packages/
↳dimscli/common/context.py to context.pyc
byte-compiling /home/dittrich/dims/envs/dimscli/lib/python2.7/site-packages/
↳dimscli/common/clientmanager.py to clientmanager.pyc
byte-compiling /home/dittrich/dims/envs/dimscli/lib/python2.7/site-packages/
↳dimscli/common/logs.py to logs.pyc
byte-compiling /home/dittrich/dims/envs/dimscli/lib/python2.7/site-packages/
↳dimscli/common/utils.py to utils.pyc
running install_egg_info
Copying python_dimscli.egg-info to /home/dittrich/dims/envs/dimscli/lib/python2.7/
↳site-packages/python_dimscli-0.0.1.dev391-py2.7.egg-info
running install_scripts
Installing dimscli script to /home/dittrich/dims/envs/dimscli/bin
```

6. Run the dimscli app like any other program, directly from the command line.

There are two ways to use dimscli.

- As a single command with command line options like other Linux commands

```
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ dimscli --
↳version
dimscli 0.0.1
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $
```

- As an interactive shell that allows you to run multiple commands in sequence within the same context (i.e., the same state, or runtime settings you invoke while in the shell) by just the program name and no arguments or options.

```
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ dimscli
defaults: {u'auth_type': 'password', u'compute_api_version': u'2', 'key':
↳None, u'database_api_version': u'1.0',
'api_timeout': None, u'baremetal_api_version': u'1', 'cacert': None, u'image_
↳api_use_tasks': False,
u'floating_ip_source': u'neutron', u'orchestration_api_version': u'1', u
↳'interface': None, u'network_api_version':
u'2.0', u'image_format': u'qcow2', u'object_api_version': u'1', u'image_api_
↳version': u'2', 'verify': True,
u'identity_api_version': u'2.0', u'volume_api_version': u'1', 'cert': None, u
↳'secgroup_source': u'neutron',
```

```

u'dns_api_version': u'2', u'disable_vendor_agent': {}
cloud cfg: {'auth_type': 'password', u'compute_api_version': u'2', u
↳ 'orchestration_api_version': u'1',
u'database_api_version': u'1.0', 'cacert': None, u'network_api_version': u'2.0
↳ ', u'image_format': u'qcow2',
u'object_api_version': u'1', u'image_api_version': u'2', 'verify': True, u
↳ 'dns_api_version': u'2',
'verbose_level': '1', 'region_name': '', 'api_timeout': None, u'baremetal_api_
↳ version': u'1', 'auth': {},
'default_domain': 'default', u'image_api_use_tasks': False, u'floating_ip_
↳ source': u'neutron', 'key': None,
'timing': False, 'deferred_help': False, u'identity_api_version': u'2.0', u
↳ 'volume_api_version': u'1',
'cert': None, u'secgroup_source': u'neutron', u'interface': None, u'disable_
↳ vendor_agent': {}}
compute API version 2, cmd group dims.compute.v2
network version 2.0 is not in supported versions 2
network API version 2.0, cmd group dims.network.v2
image API version 2, cmd group dims.image.v2
volume API version 1, cmd group dims.volume.v1
identity API version 2.0, cmd group dims.identity.v2
object_store API version 1, cmd group dims.object_store.v1
(dimscli) help

Shell commands (type help <topic>):
=====
cmdenvironment  edit  hi      l   list  pause  r   save  shell      show
ed              help  history li  load  py    run  set    shortcuts

Undocumented commands:
=====
EOF eof exit q quit

Application commands (type help <topic>):
=====
aggregate add host      host show      role list
aggregate create        ip fixed add   role remove
aggregate delete        ip fixed remove role show
aggregate list          ip floating add security group create
aggregate remove host   ip floating create security group delete
aggregate set           ip floating delete security group list
aggregate show          ip floating list security group rule create
catalog list            ip floating pool list security group rule delete
catalog show            ip floating remove security group rule list
command list            keypair create security group set
complete               keypair delete security group show
configuration show      keypair list   server create
console log show        keypair show   server delete
console url show        module list     server image create
container create         network create  server list
container delete         network delete  server reboot
container list           network list    server rebuild
container save           network set     server set
container show           network show    server show
endpoint create          object create   server ssh
endpoint delete          object delete   service create
endpoint list            object list     service delete
endpoint show            object save     service list

```

```

extension list      object show      service show
flavor create       project create   token issue
flavor delete       project delete   token revoke
flavor list         project list     user create
flavor set          project set      user delete
flavor show         project show     user list
flavor unset        role add         user role list
help                role create     user set
host list           role delete     user show

(dimscli) exit
END return value: 0
[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $

```

Command Structure

The `dimscli` shell follows the `openstack` client in the manner in which commands are to be constructed. See the [Openstack Command Structure](#) page for details. To quote:

Commands consist of an object described by one or more words followed by an action. Commands that require two objects have the primary object ahead of the action and the secondary object after the action. Any positional arguments identifying the objects shall appear in the same order as the objects. In badly formed English it is expressed as “(Take) object1 (and perform) action (using) object2 (to it).”

```
<object-1> <action> <object-2>
```

Examples:

```

$ group add user <group> <user>

$ volume type list    # 'volume type' is a two-word single object

```

Completing commands in `dimscli`

The initial implementation of `dimscli` ported from the `openstackclient` code base does not have much actual code underlying it, though the scaffolding of `openstackclient` and many of its defined modules are currently configured in the code. You can see the modules that are not there by simply asking for `dimscli --help` and noting the errors (and what they point to, which indicates which code you need to seek out to use and/or replace.)

```

[dimscli] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop) $ dimscli --help
defaults: {u'auth_type': 'password', u'compute_api_version': u'2', 'key': None, u
↪ 'database_api_version': u'1.0', 'api_timeout': None, u'baremetal_api_version': u'1',
↪ 'cacert': None, u'image_api_use_tasks
': False, u'floating_ip_source': u'neutron', u'orchestration_api_version': u'1', u
↪ 'interface': None, u'network_api_version': u'2.0', u'image_format': u'qcow2', u
↪ 'object_api_version': u'1', u'image_api_ve
rsion': u'2', 'verify': True, u'identity_api_version': u'2.0', u'volume_api_version':
↪ u'1', 'cert': None, u'secgroup_source': u'neutron', u'dns_api_version': u'2', u
↪ 'disable_vendor_agent': {}}
cloud cfg: {'auth_type': 'password', u'compute_api_version': u'2', u'orchestration_
↪ api_version': u'1', u'database_api_version': u'1.0', 'cacert': None, u'network_api_
↪ version': u'2.0', u'image_format': u'
qcow2', u'object_api_version': u'1', u'image_api_version': u'2', 'verify': True, u
↪ 'dns_api_version': u'2', 'verbose_level': '1', 'region_name': '', 'api_timeout':
↪ None, u'baremetal_api_version': u'1', 'a

```

```
uth': {}, 'default_domain': 'default', u'image_api_use_tasks': False, u'floating_ip_
↪source': u'neutron', 'key': None, 'timing': False, 'deferred_help': True, u
↪'identity_api_version': u'2.0', u'volume_api
_version': u'1', 'cert': None, u'secgroup_source': u'neutron', u'interface': None, u
↪'disable_vendor_agent': {}})
compute API version 2, cmd group dims.compute.v2
network version 2.0 is not in supported versions 2
network API version 2.0, cmd group dims.network.v2
image API version 2, cmd group dims.image.v2
volume API version 1, cmd group dims.volume.v1
identity API version 2.0, cmd group dims.identity.v2
object_store API version 1, cmd group dims.object_store.v1
usage: dimscli [--version] [-v] [--log-file LOG_FILE] [-q] [-h] [--debug]
        [--os-cloud <cloud-config-name>]
        [--os-region-name <auth-region-name>]
        [--os-cacert <ca-bundle-file>] [--verify | --insecure]
        [--os-default-domain <auth-domain>]
...

--os-object-api-version <object-api-version>
                        Object API version, default=1 (Env:
                        OS_OBJECT_API_VERSION)

Commands:
Could not load EntryPoint.parse('aggregate_add_host = dimscli.compute.v2.
↪aggregate:AddAggregateHost')
Could not load EntryPoint.parse('aggregate_create = dimscli.compute.v2.
↪aggregate:CreateAggregate')
Could not load EntryPoint.parse('aggregate_delete = dimscli.compute.v2.
↪aggregate>DeleteAggregate')
Could not load EntryPoint.parse('aggregate_list = dimscli.compute.v2.
↪aggregate:ListAggregate')
Could not load EntryPoint.parse('aggregate_remove_host = dimscli.compute.v2.
↪aggregate:RemoveAggregateHost')
Could not load EntryPoint.parse('aggregate_set = dimscli.compute.v2.
↪aggregate:SetAggregate')
Could not load EntryPoint.parse('aggregate_show = dimscli.compute.v2.
↪aggregate:ShowAggregate')
Could not load EntryPoint.parse('catalog_list = dimscli.identity.v2_0.
↪catalog:ListCatalog')
Could not load EntryPoint.parse('catalog_show = dimscli.identity.v2_0.
↪catalog:ShowCatalog')
Could not load EntryPoint.parse('command_list = dimscli.common.module:ListCommand')
complete      print bash completion command
Could not load EntryPoint.parse('configuration_show = dimscli.common.
↪configuration:ShowConfiguration')
Could not load EntryPoint.parse('console_log_show = dimscli.compute.v2.
↪console:ShowConsoleLog')
Could not load EntryPoint.parse('console_url_show = dimscli.compute.v2.
↪console:ShowConsoleURL')
Could not load EntryPoint.parse('container_create = dimscli.object.v1.
↪container:CreateContainer')
Could not load EntryPoint.parse('container_delete = dimscli.object.v1.
↪container>DeleteContainer')
Could not load EntryPoint.parse('container_list = dimscli.object.v1.
↪container:ListContainer')
Could not load EntryPoint.parse('container_save = dimscli.object.v1.
↪container:SaveContainer')
```

```

Could not load EntryPoint.parse('container_show = dimscli.object.v1.
↳container:ShowContainer')
Could not load EntryPoint.parse('endpoint_create = dimscli.identity.v2_0.
↳endpoint:CreateEndpoint')
Could not load EntryPoint.parse('endpoint_delete = dimscli.identity.v2_0.
↳endpoint>DeleteEndpoint')
Could not load EntryPoint.parse('endpoint_list = dimscli.identity.v2_0.
↳endpoint>ListEndpoint')
Could not load EntryPoint.parse('endpoint_show = dimscli.identity.v2_0.
↳endpoint>ShowEndpoint')
Could not load EntryPoint.parse('extension_list = dimscli.common.
↳extension>ListExtension')
Could not load EntryPoint.parse('flavor_create = dimscli.compute.v2.
↳flavor>CreateFlavor')
Could not load EntryPoint.parse('flavor_delete = dimscli.compute.v2.
↳flavor>DeleteFlavor')
Could not load EntryPoint.parse('flavor_list = dimscli.compute.v2.flavor:ListFlavor')
Could not load EntryPoint.parse('flavor_set = dimscli.compute.v2.flavor:SetFlavor')
Could not load EntryPoint.parse('flavor_show = dimscli.compute.v2.flavor:ShowFlavor')
Could not load EntryPoint.parse('flavor_unset = dimscli.compute.v2.flavor:UnsetFlavor
↳')
    help                print detailed help for another command
Could not load EntryPoint.parse('host_list = dimscli.compute.v2.host:ListHost')
Could not load EntryPoint.parse('host_show = dimscli.compute.v2.host:ShowHost')
Could not load EntryPoint.parse('ip_fixed_add = dimscli.compute.v2.fixedip:AddFixedIP
↳')
Could not load EntryPoint.parse('ip_fixed_remove = dimscli.compute.v2.
↳fixedip:RemoveFixedIP')
Could not load EntryPoint.parse('ip_floating_add = dimscli.compute.v2.
↳floatingip:AddFloatingIP')
Could not load EntryPoint.parse('ip_floating_create = dimscli.compute.v2.
↳floatingip>CreateFloatingIP')
Could not load EntryPoint.parse('ip_floating_delete = dimscli.compute.v2.
↳floatingip>DeleteFloatingIP')
Could not load EntryPoint.parse('ip_floating_list = dimscli.compute.v2.
↳floatingip>ListFloatingIP')
Could not load EntryPoint.parse('ip_floating_pool_list = dimscli.compute.v2.
↳floatingippool>ListFloatingIPPool')
Could not load EntryPoint.parse('ip_floating_remove = dimscli.compute.v2.
↳floatingip:RemoveFloatingIP')
Could not load EntryPoint.parse('keypair_create = dimscli.compute.v2.
↳keypair>CreateKeypair')
Could not load EntryPoint.parse('keypair_delete = dimscli.compute.v2.
↳keypair>DeleteKeypair')
Could not load EntryPoint.parse('keypair_list = dimscli.compute.v2.keypair:ListKeypair
↳')
Could not load EntryPoint.parse('keypair_show = dimscli.compute.v2.keypair:ShowKeypair
↳')
Could not load EntryPoint.parse('module_list = dimscli.common.module:ListModule')
Could not load EntryPoint.parse('network_create = dimscli.network.v2.
↳network>CreateNetwork')
Could not load EntryPoint.parse('network_delete = dimscli.network.v2.
↳network>DeleteNetwork')
Could not load EntryPoint.parse('network_list = dimscli.network.v2.network:ListNetwork
↳')
Could not load EntryPoint.parse('network_set = dimscli.network.v2.network:SetNetwork')
Could not load EntryPoint.parse('network_show = dimscli.network.v2.network:ShowNetwork
↳')

```

```
Could not load EntryPoint.parse('object_create = dimscli.object.v1.object:CreateObject
↳')
Could not load EntryPoint.parse('object_delete = dimscli.object.v1.object:DeleteObject
↳')
Could not load EntryPoint.parse('object_list = dimscli.object.v1.object:ListObject')
Could not load EntryPoint.parse('object_save = dimscli.object.v1.object:SaveObject')
Could not load EntryPoint.parse('object_show = dimscli.object.v1.object:ShowObject')
Could not load EntryPoint.parse('project_create = dimscli.identity.v2_0.
↳project:CreateProject')
Could not load EntryPoint.parse('project_delete = dimscli.identity.v2_0.
↳project:DeleteProject')
Could not load EntryPoint.parse('project_list = dimscli.identity.v2_0.
↳project:ListProject')
Could not load EntryPoint.parse('project_set = dimscli.identity.v2_0.
↳project:SetProject')
Could not load EntryPoint.parse('project_show = dimscli.identity.v2_0.
↳project:ShowProject')
Could not load EntryPoint.parse('role_add = dimscli.identity.v2_0.role:AddRole')
Could not load EntryPoint.parse('role_create = dimscli.identity.v2_0.role:CreateRole')
Could not load EntryPoint.parse('role_delete = dimscli.identity.v2_0.role:DeleteRole')
Could not load EntryPoint.parse('role_list = dimscli.identity.v2_0.role:ListRole')
Could not load EntryPoint.parse('role_remove = dimscli.identity.v2_0.role:RemoveRole')
Could not load EntryPoint.parse('role_show = dimscli.identity.v2_0.role:ShowRole')
Could not load EntryPoint.parse('security_group_create = dimscli.compute.v2.security_
↳group:CreateSecurityGroup')
Could not load EntryPoint.parse('security_group_delete = dimscli.compute.v2.security_
↳group:DeleteSecurityGroup')
Could not load EntryPoint.parse('security_group_list = dimscli.compute.v2.security_
↳group:ListSecurityGroup')
Could not load EntryPoint.parse('security_group_rule_create = dimscli.compute.v2.
↳security_group:CreateSecurityGroupRule')
Could not load EntryPoint.parse('security_group_rule_delete = dimscli.compute.v2.
↳security_group:DeleteSecurityGroupRule')
Could not load EntryPoint.parse('security_group_rule_list = dimscli.compute.v2.
↳security_group:ListSecurityGroupRule')
Could not load EntryPoint.parse('security_group_set = dimscli.compute.v2.security_
↳group:SetSecurityGroup')
Could not load EntryPoint.parse('security_group_show = dimscli.compute.v2.security_
↳group:ShowSecurityGroup')
Could not load EntryPoint.parse('server_create = dimscli.compute.v2.
↳server:CreateServer')
Could not load EntryPoint.parse('server_delete = dimscli.compute.v2.
↳server:DeleteServer')
Could not load EntryPoint.parse('server_image_create = dimscli.compute.v2.
↳server:CreateServerImage')
Could not load EntryPoint.parse('server_list = dimscli.compute.v2.server:ListServer')
Could not load EntryPoint.parse('server_reboot = dimscli.compute.v2.
↳server:RebootServer')
Could not load EntryPoint.parse('server_rebuild = dimscli.compute.v2.
↳server:RebuildServer')
Could not load EntryPoint.parse('server_set = dimscli.compute.v2.server:SetServer')
Could not load EntryPoint.parse('server_show = dimscli.compute.v2.server:ShowServer')
Could not load EntryPoint.parse('server_ssh = dimscli.compute.v2.server:SshServer')
Could not load EntryPoint.parse('service_create = dimscli.identity.v2_0.
↳service:CreateService')
Could not load EntryPoint.parse('service_delete = dimscli.identity.v2_0.
↳service:DeleteService')
Could not load EntryPoint.parse('service_list = dimscli.identity.v2_0.
↳service:ListService')
```

```

Could not load EntryPoint.parse('service_show = dimsccli.identity.v2_0.
↳service:ShowService')
Could not load EntryPoint.parse('token_issue = dimsccli.identity.v2_0.token:IssueToken
↳')
Could not load EntryPoint.parse('token_revoke = dimsccli.identity.v2_0.
↳token:RevokeToken')
Could not load EntryPoint.parse('user_create = dimsccli.identity.v2_0.user:CreateUser')
Could not load EntryPoint.parse('user_delete = dimsccli.identity.v2_0.user:DeleteUser')
Could not load EntryPoint.parse('user_list = dimsccli.identity.v2_0.user:ListUser')
Could not load EntryPoint.parse('user_role_list = dimsccli.identity.v2_0.
↳role:ListUserRole')
Could not load EntryPoint.parse('user_set = dimsccli.identity.v2_0.user:SetUser')
Could not load EntryPoint.parse('user_show = dimsccli.identity.v2_0.user:ShowUser')
END return value: 1
[dimsccli] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop) $

```

Using the last error message above as an example, there needs to be a module named `$GIT/python-dimsccli/dimsccli/identity/v2_0/user.py` with a class `ShowUser`. Look in the `python-openstack/openstack/identity/v2_0/` directory for their `user.py` and build off that example.

Attention: Clone the `python-openstackclient` repo using `git clone https://git.openstack.org/openstack/python-openstackclient` and see the `cliff` documentation, [Section Exploring the Demo App](#), for how this works.

Attention: See the file `$GIT/python-dimsccli/README.rst` for more documentation produced during initial creation of the `openstackclient` fork of `dimsccli`.

`cliff` supports list formatting in tables, CSV, JSON, etc., but not in shell format. That is only supported by the `ShowOne` class, which is not what we want for producing a set of variables for insertion into shell environments.

```

[dimseenv] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop*) $ dimsccli list nodes
+-----+-----+
| Node           | Address       |
+-----+-----+
| b52            | 10.86.86.7    |
| consul-breathe | 10.142.29.117 |
| consul-echoes  | 10.142.29.116 |
| consul-seamus  | 10.142.29.120 |
| dimsdemo1     | 10.86.86.2    |
| dimsdev1      | 10.86.86.5    |
| dimsdev2      | 10.86.86.5    |
| four          | 192.168.0.101 |
+-----+-----+

```

```

[dimseenv] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop*) $ dimsccli list_
↳nodes -f csv
"Node", "Address"
"b52", "10.86.86.7"
"consul-breathe", "10.142.29.117"
"consul-echoes", "10.142.29.116"
"consul-seamus", "10.142.29.120"
"dimsdemo1", "10.86.86.2"
"dimsdev1", "10.86.86.5"

```

```
"dimsdev2", "10.86.86.5"  
"four", "192.168.0.101"
```

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop*) $ dimscli list_  
↪nodes -f json  
[{"Node": "b52", "Address": "10.86.86.7"}, {"Node": "consul-breathe", "Address": "10.  
↪142.29.117"}, {"Node": "consul-echoes", "Address": "10.142.29.116"}, {"Node":  
↪"consul-seamus", "Address": "10.142.29.120"}, {"Node": "dimsdemo1", "Address": "10.  
↪86.86.2"}, {"Node": "dimsdev1", "Address": "10.86.86.5"}, {"Node": "dimsdev2",  
↪"Address": "10.86.86.5"}, {"Node": "four", "Address": "192.168.0.101"}]  
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop*) $ dimscli list_  
↪nodes -f json | python -m json.tool  
[  
  {  
    "Address": "10.86.86.7",  
    "Node": "b52"  
  },  
  {  
    "Address": "10.142.29.117",  
    "Node": "consul-breathe"  
  },  
  {  
    "Address": "10.142.29.116",  
    "Node": "consul-echoes"  
  },  
  {  
    "Address": "10.142.29.120",  
    "Node": "consul-seamus"  
  },  
  {  
    "Address": "10.86.86.2",  
    "Node": "dimsdemo1"  
  },  
  {  
    "Address": "10.86.86.5",  
    "Node": "dimsdev1"  
  },  
  {  
    "Address": "10.86.86.5",  
    "Node": "dimsdev2"  
  },  
  {  
    "Address": "192.168.0.101",  
    "Node": "four"  
  }  
]
```

To produce the list in the form of shell variables, we need to create a custom formatter and load it into the `dimscli` shell via `Stevedore`.

After adding the new formatter, it is possible to extract the list of nodes registered with Consul and produce a set of variable declarations from the list.

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimscli (develop*) $ dimscli list_  
↪nodes -f shell  
b52="10.86.86.7"  
consul_breathe="10.142.29.117"  
consul_echoes="10.142.29.116"
```



```
consul_seamus="10.142.29.120"
dimsdemol="10.86.86.2"
dimsdev1="10.86.86.5"
dimsdev2="10.86.86.5"
four="192.168.0.101"
```

In practice, you may wish to insert these as variables in the shell's `set` using the `eval` statement for use when invoking shell commands:

```
[dimesenv] dittrich@dimsdemol:~/dime/git/python-dimecli (develop*) $ eval $(dimecli_
↪list nodes -f shell --prefix=DIMS_)
[dimesenv] dittrich@dimsdemol:~/dime/git/python-dimecli (develop*) $ set | grep DIMS_
DIMS_REV=unspecified
DIMS_VERSION='1.6.124 (dime-ci-utils)'
DIMS_b52=10.86.86.7
DIMS_consul_breathe=10.142.29.117
DIMS_consul_echoes=10.142.29.116
DIMS_consul_seamus=10.142.29.120
DIMS_dime-demol=10.86.86.2
DIMS_dime-dev2=10.86.86.5
DIMS_four=192.168.0.101
    echo "REV:      $DIMS_REV";
    echo "[dime-ci-utils version $(version) (rev $DIMS_REV)]";
    echo "$PROGRAM $DIMS_VERSION";
    echo "$BASE $DIMS_VERSION";
```

Adding New Columns to Output

Say we want to also include the Consul status, to help determine which node is currently the *Leader* in a cluster, which are a *Peer* in the cluster, and which are simply an *Agent* that is proxying to the cluster.

The changes to existing code to affect this new feature are shown here:

```
commit caab2d05274898878e1123bd337b431c8d2f2a8e
Author: Dave Dittrich <dittrich@u.washington.edu>
Date: Sat Jan 2 12:53:56 2016 -0800

    Add Consul node status to 'nodes list' output

diff --git a/dimecli/list.py b/dimecli/list.py
index 45acdda..3893b10 100644
--- a/dimecli/list.py
+++ b/dimecli/list.py
@@ -26,9 +26,35 @@ class Nodes(Lister):

    log = logging.getLogger(__name__)

+    def get_node_status(self):
+        """
+        Determine the status from Consul
+
+        :return: None
+        """
+        self.leaderDict = dict(zip(['Address', 'Port'],
+                                   self.consul.status.leader().split(":")))
+        self.peersDictList = [dict(zip(['Address', 'Port'], p.split(":")))]
```

```

+                 for p in self.consul.status.peers()]
+
+     def status(self, address):
+         """
+         Determine node status as returned from Consul.
+
+         :param address: IP address to check
+         :return: One of: "Leader", "Peer", or "Agent"
+         """
+         if address in self.leaderDict.values():
+             return "Leader"
+         elif address in [p['Address'] for p in self.peersDictList]:
+             return "Peer"
+         else:
+             return "Agent"
+
+     def take_action(self, parsed_args):
-         consul = consulate.Consul()
-         nodes = consul.catalog.nodes()
-         columns = ('Node', 'Address')
-         data = ((node['Node'], node['Address']) for node in nodes)
+         self.consul = consulate.Consul()
+         nodes = self.consul.catalog.nodes()
+         self.get_node_status()
+         columns = ('Node', 'Address', 'Status')
+         data = ((node['Node'], node['Address'], self.status(node['Address'])) for
+↪node in nodes)
+         return (columns, data)

```

```

[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop*) $ dimsccli nodes list
+-----+-----+-----+
| Node      | Address      | Status |
+-----+-----+-----+
| b52       | 10.86.86.2   | Agent  |
| breathe   | 10.142.29.117 | Leader |
| dimsdemo1 | 10.86.86.3   | Agent  |
| echoes    | 10.142.29.116 | Peer   |
| seamus    | 10.142.29.120 | Peer   |
+-----+-----+-----+
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop*) $ dimsccli nodes_
↪list -f csv
"Node","Address","Status"
"b52","10.86.86.2","Agent"
"breathe","10.142.29.117","Leader"
"dimsdemo1","10.86.86.3","Agent"
"echoes","10.142.29.116","Peer"
"seamus","10.142.29.120","Peer"
[dimsenv] dittrich@dimsdemo1:~/dims/git/python-dimsccli (develop*) $ dimsccli nodes_
↪list -f json | python -mjson.tool
[
  {
    "Address": "10.86.86.2",
    "Node": "b52",
    "Status": "Agent"
  },
  {
    "Address": "10.142.29.117",
    "Node": "breathe",

```

```

        "Status": "Leader"
    },
    {
        "Address": "10.86.86.3",
        "Node": "dimsdemo1",
        "Status": "Agent"
    },
    {
        "Address": "10.142.29.116",
        "Node": "echoes",
        "Status": "Peer"
    },
    {
        "Address": "10.142.29.120",
        "Node": "seamus",
        "Status": "Peer"
    }
]

```

If we wish to turn a subset of this table into variables, using the `shell` output feature added above, we need to select a pair of columns (to map to *Variable=Value* in the output). The results could then be used in Ansible playbooks, shell scripts, selecting color for nodes in a graph, or any number of other purposes.

```

[dimsenv] dittrich@dimsdemo1:~/dms/git/python-dimscli (develop*) $ dimscli nodes_
↪list --column Node --column Status -f shell
b52="Agent"
breathe="Leader"
dimsdemo1="Agent"
echoes="Peer"
seamus="Peer"

```

Adding New Commands

In this example, we will add a new command `ansible` with a subcommand `execute` that will use Ansible's [Python API](#) (specifically the `ansible.runner.Runner` class) to execute arbitrary commands on hosts via Ansible.

Note: What is being demonstrated here is adding a new subcommand to the `dimscli` repo directly. It is also possible to add a new command from a module in another repo using [Stevedore](#).

Here are the changes that implement this new command:

```

commit eccf3af707aac5a13144580bfbf548b45616d49f
Author: Dave Dittrich <dittrich@u.washington.edu>
Date:   Fri Jan 1 20:34:42 2016 -0800

    Add 'ansible execute' command

diff --git a/dimscli/dimsansible/__init__.py b/dimscli/dimsansible/__init__.py
new file mode 100644
index 0000000..e69de29
diff --git a/dimscli/dimsansible/ansiblerunner.py b/dimscli/dimsansible/ansiblerunner.
↪py
new file mode 100644
index 0000000..68cd3ea

```

```
--- /dev/null
+++ b/dimsccli/dimsansible/ansiblerunner.py
@@ -0,0 +1,61 @@
+#!/usr/bin/python
+
+import sys
+import logging
+
+from cliff.lister import Lister
+from ansible.runner import Runner
+
+HOST_LIST = "/etc/ansible/hosts"
+CMD = "/usr/bin/uptime"
+
+class Execute(Lister):
+    """Execute a command via Ansible and return a list of results.
+
+    """
+
+    log = logging.getLogger(__name__)
+
+    def get_parser(self, prog_name):
+        parser = super(Execute, self).get_parser(prog_name)
+        parser.add_argument(
+            "--host-list",
+            metavar="<host-list>",
+            default=HOST_LIST,
+            help="Hosts file (default: {}).".format(HOST_LIST),
+        )
+        parser.add_argument(
+            "--program",
+            metavar="<program>",
+            default=CMD,
+            help="Program to run (default: {}).".format(CMD),
+        )
+        return parser
+
+    def take_action(self, parsed_args):
+
+        results = Runner(
+            host_list=parsed_args.host_list,
+            pattern='*',
+            forks=10,
+            module_name='command',
+            module_args=parsed_args.program,
+        ).run()
+
+        if results is None:
+            print "No hosts found"
+            sys.exit(1)
+
+        outtable = []
+
+        for (hostname, result) in results['contacted'].items():
+            if not 'failed' in result:
+                outtable.append((hostname, 'GOOD', result['stdout']))
+            elif 'failed' in result:
+                outtable.append((hostname, 'FAIL', result['msg']))
```

```

+         for (hostname, result) in results['dark'].items():
+             outtable.append((hostname, 'DARK', result['msg']))
+
+         column_names = ('Host', 'Status', 'Results')
+
+         return column_names, outtable
diff --git a/setup.cfg b/setup.cfg
index 14f6ce7..9571d4f 100644
--- a/setup.cfg
+++ b/setup.cfg
@@ -37,6 +37,7 @@ dims.cli =
     files_list = dimscli.list:Files
     nodes_list = dimscli.list:Nodes
     show_file = dimscli.show:File
+    ansible_execute = dimscli.dimsansible.ansiblerunner:Execute

cliff.formatter.list =
    shell = dimscli.formatters.shell:DIMSShellFormatter

```

Here is what the command can do (as seen in the --help output).

```

[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ dimscli_
↪ansible execute --help
usage: dimscli ansible execute [-h]
                               [-f {csv,html,json,json,shell,table,value,yaml,yaml}]
                               [-c COLUMN] [--prefix PREFIX]
                               [--max-width <integer>] [--noindent]
                               [--quote {all,minimal,none,nonnumeric}]
                               [--host-list <host-list>] [--program <program>]

```

Execute a command via Ansible and return a list of results.

optional arguments:

```

-h, --help                show this help message and exit
--host-list <host-list>   Hosts file (default: /etc/ansible/hosts)
--program <program>       Program to run (default: /usr/bin/uptime)

```

output formatters:

output formatter options

```

-f {csv,html,json,json,shell,table,value,yaml,yaml}, --format {csv,html,json,json,
↪shell,table,value,yaml,yaml}
                               the output format, defaults to table
-c COLUMN, --column COLUMN
                               specify the column(s) to include, can be repeated

```

shell formatter:

a format a UNIX shell can parse (variable="value")

```

--prefix PREFIX            add a prefix to all variable names

```

table formatter:

```

--max-width <integer>
                               Maximum display width, 0 to disable

```

json formatter:

```

--noindent                whether to disable indenting the JSON

```

CSV Formatter:

```
--quote {all,minimal,none,nonnumeric}
           when to include quotes, defaults to nonnumeric
```

The script defaults to using the standard Ansible `/etc/ansible/hosts` file to get its inventory. In this case, the DIMS `$GIT/ansible-inventory/development` file was copied to the default location. Using this file to execute the default command `/usr/bin/uptime` on the defined development hosts results in the following:

```
[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ dimscli
↪ansible execute
+-----+-----+-----+
↪-----+
| Host                                     | Status | Results                                     |
↪-----+-----+-----+
↪-----+
| linda-vm1.prisem.washington.edu         | GOOD   | 18:31:22 up 146 days, 8:27, 1
↪user, load average: 0.00, 0.01, 0.05 |
| ul2-dev-ws-1.prisem.washington.edu      | GOOD   | 18:31:21 up 146 days, 8:27, 1
↪user, load average: 0.00, 0.01, 0.05 |
| hub.prisem.washington.edu               | GOOD   | 02:31:22 up 128 days, 8:42, 1
↪user, load average: 0.00, 0.01, 0.05 |
| floyd2-p.prisem.washington.edu          | GOOD   | 18:31:21 up 20 days, 56 min, 1
↪user, load average: 0.02, 0.04, 0.05 |
| ul2-dev-svr-1.prisem.washington.edu     | GOOD   | 18:31:22 up 142 days, 11:22, 1
↪user, load average: 0.00, 0.01, 0.05 |
+-----+-----+-----+
↪-----+
```

Using the `--program` command line option, a different command can be run:

```
[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ dimscli
↪ansible execute --program "ip addr"
+-----+-----+-----+
↪-----+
| Host                                     | Status | Results                                     |
↪-----+-----+-----+
↪-----+
| linda-vm1.prisem.washington.edu         | GOOD   | 1: lo: <LOOPBACK,UP,LOWER_UP> mtu
↪65536 qdisc noqueue state UNKNOWN
|                                     |        | link/loopback 00:00:00:00:00:00
↪brd 00:00:00:00:00:00
|                                     |        | inet 127.0.0.1/8 scope host lo
↪
|                                     |        | valid_lft forever preferred_
↪lft forever
|                                     |        | 2: eth0: <BROADCAST,MULTICAST,UP,
↪LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
|                                     |        | link/ether 08:00:27:3b:3a:65 brd
↪ff:ff:ff:ff:ff:ff
|                                     |        | inet 10.0.2.15/24 brd 10.0.2.255
↪scope global eth0
|                                     |        | valid_lft forever preferred_
↪lft forever
|                                     |        | 3: eth1: <BROADCAST,MULTICAST,UP,
↪LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
|                                     |        |
```

			link/ether 08:00:27:36:2b:2c brd
↪ff:ff:ff:ff:ff:ff			
			inet 192.168.88.11/24 brd 192.
↪168.88.255 scope global eth1			
			valid_lft forever preferred
↪lft forever			
ul2-dev-svr-1.prisem.washington.edu	GOOD	1: lo: <LOOPBACK,UP,LOWER_UP> mtu	
↪65536 qdisc noqueue state UNKNOWN			
			link/loopback 00:00:00:00:00:00
↪brd 00:00:00:00:00:00			
			inet 127.0.0.1/8 scope host lo
↪			
			valid_lft forever preferred
↪lft forever			
			inet6 ::1/128 scope host
↪			
			valid_lft forever preferred
↪lft forever			
		2: eth0: <BROADCAST,MULTICAST,UP,	
↪LOWER_UP> mtu 1500 qdisc pfifo_fast	state UP	qlen 1000	
			link/ether 08:00:27:38:db:8c brd
↪ff:ff:ff:ff:ff:ff			
			inet 10.0.2.15/24 brd 10.0.2.255
↪scope global eth0			
			valid_lft forever preferred
↪lft forever			
			inet6 fe80::a00:27ff:fe38:db8c/
↪64 scope link			
			valid_lft forever preferred
↪lft forever			
		3: eth1: <BROADCAST,MULTICAST,UP,	
↪LOWER_UP> mtu 1500 qdisc pfifo_fast	state UP	qlen 1000	
			link/ether 08:00:27:e7:80:52 brd
↪ff:ff:ff:ff:ff:ff			
			inet 192.168.88.13/24 brd 192.
↪168.88.255 scope global eth1			
			valid_lft forever preferred
↪lft forever			
			inet6 fe80::a00:27ff:fee7:8052/
↪64 scope link			
			valid_lft forever preferred
↪lft forever			
hub.prisem.washington.edu	GOOD	1: lo: <LOOPBACK,UP,LOWER_UP> mtu	
↪65536 qdisc noqueue state UNKNOWN	group default		
			link/loopback 00:00:00:00:00:00
↪brd 00:00:00:00:00:00			
			inet 127.0.0.1/8 scope host lo
↪			
			valid_lft forever preferred
↪lft forever			
			inet6 ::1/128 scope host
↪			
			valid_lft forever preferred
↪lft forever			
		2: eth0: <BROADCAST,MULTICAST,UP,	
↪LOWER_UP> mtu 1500 qdisc pfifo_fast	state UP	group default qlen 1000	
			link/ether 08:00:27:9c:f8:95 brd
↪ff:ff:ff:ff:ff:ff			

			inet 10.0.2.15/24 brd 10.0.2.255	
↪scope global eth0				
			valid_lft forever preferred_	
↪lft forever				
			inet6 fe80::a00:27ff:fe9c:f895/	
↪64 scope link				
			valid_lft forever preferred_	
↪lft forever				
			3: eth1: <BROADCAST,MULTICAST,UP,	
↪LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000				
			link/ether 08:00:27:28:63:2a brd	
↪ff:ff:ff:ff:ff:ff				
			inet 192.168.88.14/24 brd 192.	
↪168.88.255 scope global eth1				
			valid_lft forever preferred_	
↪lft forever				
			inet6 fe80::a00:27ff:fe28:632a/	
↪64 scope link				
			valid_lft forever preferred_	
↪lft forever				
			4: docker0: <BROADCAST,MULTICAST,UP,	
↪LOWER_UP> mtu 1500 qdisc noqueue state UP group default				
			link/ether 56:84:7a:fe:97:99 brd	
↪ff:ff:ff:ff:ff:ff				
			inet 172.17.42.1/16 scope global	
↪docker0				
			valid_lft forever preferred_	
↪lft forever				
			inet6 fe80::5484:7aff:fefe:9799/	
↪64 scope link				
			valid_lft forever preferred_	
↪lft forever				
			22: veth6dc6dd5: <BROADCAST,	
↪MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default				
↪				
			link/ether 8e:d6:f5:66:fb:88 brd	
↪ff:ff:ff:ff:ff:ff				
			inet6 fe80::8cd6:f5ff:fe66:fb88/	
↪64 scope link				
			valid_lft forever preferred_	
↪lft forever				
			42: vethdc35259: <BROADCAST,	
↪MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default				
↪				
			link/ether 46:c3:87:32:83:a1 brd	
↪ff:ff:ff:ff:ff:ff				
			inet6 fe80::44c3:87ff:fe32:83a1/	
↪64 scope link				
			valid_lft forever preferred_	
↪lft forever				
			1: lo: <LOOPBACK,UP,LOWER_UP> mtu	
↪floyd2-p.prisem.washington.edu 65536 qdisc noqueue state UNKNOWN				
			link/loopback 00:00:00:00:00:00	
↪brd 00:00:00:00:00:00				
			inet 127.0.0.1/8 scope host lo	
↪				
			valid_lft forever preferred_	
↪lft forever				


```

| | | 2: eth0: <BROADCAST,MULTICAST,UP,
↳LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000 |
| | | link/ether 52:54:00:17:19:9a brd
↳ff:ff:ff:ff:ff:ff |
| | | inet 172.22.29.175/24 brd 172.22.
↳29.255 scope global eth0 |
| | | valid_lft forever preferred_
↳lft forever |
| | | 3: eth1: <BROADCAST,MULTICAST> mtu
↳1500 qdisc noop state DOWN qlen 1000 |
| | | link/ether 52:54:00:85:34:b7 brd
↳ff:ff:ff:ff:ff:ff |
| ul2-dev-ws-1.prisem.washington.edu | GOOD | 1: lo: <LOOPBACK,UP,LOWER_UP> mtu
↳65536 qdisc noqueue state UNKNOWN |
| | | link/loopback 00:00:00:00:00:00
↳brd 00:00:00:00:00:00 |
| | | inet 127.0.0.1/8 scope host lo
↳ |
| | | valid_lft forever preferred_
↳lft forever |
| | | 2: eth0: <BROADCAST,MULTICAST,UP,
↳LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000 |
| | | link/ether 08:00:27:07:6b:00 brd
↳ff:ff:ff:ff:ff:ff |
| | | inet 10.0.2.15/24 brd 10.0.2.255
↳scope global eth0 |
| | | valid_lft forever preferred_
↳lft forever |
| | | 3: eth1: <BROADCAST,MULTICAST,UP,
↳LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000 |
| | | link/ether 08:00:27:75:a0:25 brd
↳ff:ff:ff:ff:ff:ff |
| | | inet 192.168.88.12/24 brd 192.
↳168.88.255 scope global eth1 |
| | | valid_lft forever preferred_
↳lft forever |
| | | 4: docker0: <BROADCAST,MULTICAST,UP,
↳LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN |
| | | link/ether 5a:cb:bd:c2:f5:82 brd
↳ff:ff:ff:ff:ff:ff |
| | | inet 172.17.42.1/16 scope global
↳docker0 |
| | | valid_lft forever preferred_
↳lft forever |
+-----+-----+-----+
↳-----+

```

```

[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ dimscli
↳ansible execute --program "cat /etc/hosts"
+-----+-----+-----+
↳-----+
| Host | Status | Results |
↳ |
↳ |
+-----+-----+-----+
↳-----+
↳-----+

```

```

| linda-vm1.prisem.washington.edu | GOOD | 127.0.0.1 | localhost |
↪
↪
| | | 127.0.1.1 | ubul2-generic |
↪
↪
| | |
↪
↪
| | | # The following lines are desirable
↪for IPv6 capable hosts
↪
| | | ::1 | ip6-localhost ip6-loopback |
↪
↪
| | | fe00::0 | ip6-localnet |
↪
↪
| | | ff00::0 | ip6-mcastprefix |
↪
↪
| | | ff02::1 | ip6-allnodes |
↪
↪
| | | ff02::2 | ip6-allrouters |
↪
↪
| | |
↪
↪
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| | | 127.0.0.1 | auth-test.prisem.
↪washington.edu manager-test.prisem.washington.edu reload-test.prisem.washington.
↪edu test5.prisem.washingto |
| ul2-dev-svr-1.prisem.washington.edu | GOOD | 127.0.0.1 | localhost |
↪
↪
| |

```



```
|
↪
↪
|
↪for IPv6 capable hosts
↪
|
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
+-----+-----+-----+
↪-----+
↪-----+
```

To run a command across the full set of ansible-compatible hosts, we can use the helper Makefile in the \$GIT/ansible-inventory repo to extract a list of *all* hosts specified in *any* inventory file to form a complete set.

Note: This helper Makefile was originally written to take a set of static inventory files and generate a set, rather than forcing someone to manually edit a file and manually combine all hosts from any file (which is error prone, tedious, difficult to remember how to do... basically impractical for a scalable solution.)

```
[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ (cd $GIT/
↪ansible-inventory; make help)
usage: make [something]
```

Where 'something' is one of:

```
help - Show this help information
all -      Default is create complete_inventory file.

inventory - Create file 'complete_inventory' with all hosts
            from any file with an '[all]' section in it.

tree -      Produce a tree listing of everything except
            'older-*' directories.

clean -      Clean up files.
```

```
[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ (cd $GIT/
↪ansible-inventory; make inventory)
echo '[all]' > complete_inventory
cat development hosts-old infrastructure Makefile prisem project | awk '\
    /^[all\]/ { echo = 1; next; }\
    /^$/      { echo = 0; }\
```

```
{ if (echo == 1) { print; } }' |\n
sort | uniq >> complete_inventory
```

Now this list can be used to run the command across the full set of hosts under Ansible control.

```
[dimscli] dittrich@dimsdemo1:ims/git/python-dimscli/dimscli (develop*) $ dimscli
↪ansible execute --host-list /home/dittrich/dims/git/ansible-inventory/complete_
↪inventory
```

Host	Status	Results
rabbitmq.prisem.washington.edu	GOOD	18:35:04 up 20 days, 1:00, 1 user, load average: 0.00, 0.04, 0.05
wellington.prisem.washington.edu	GOOD	18:35:06 up 146 days, 8:43, 1 user, load average: 0.43, 0.64, 0.43
hub.prisem.washington.edu	GOOD	02:35:02 up 128 days, 8:46, 1 user, load average: 0.11, 0.06, 0.05
git.prisem.washington.edu	GOOD	18:35:03 up 146 days, 8:30, 2 users, load average: 0.18, 0.07, 0.06
time.prisem.washington.edu	GOOD	18:35:04 up 20 days, 1:00, 2 users, load average: 0.06, 0.13, 0.13
jira-int.prisem.washington.edu	GOOD	18:35:03 up 146 days, 8:30, 2 users, load average: 0.18, 0.07, 0.06
ul2-dev-ws-1.prisem.washington.edu	GOOD	18:35:05 up 146 days, 8:30, 1 user, load average: 0.01, 0.02, 0.05
sso.prisem.washington.edu	GOOD	18:35:05 up 146 days, 8:30, 1 user, load average: 0.00, 0.02, 0.05
lapp-int.prisem.washington.edu	GOOD	18:35:02 up 146 days, 8:31, 2 users, load average: 0.16, 0.05, 0.06
foswiki-int.prisem.washington.edu	GOOD	18:35:03 up 146 days, 8:31, 1 user, load average: 0.00, 0.01, 0.05
ul2-dev-svr-1.prisem.washington.edu	GOOD	18:35:03 up 142 days, 11:26, 1 user, load average: 0.03, 0.04, 0.05
linda-vm1.prisem.washington.edu	GOOD	18:35:05 up 146 days, 8:31, 1 user, load average: 0.13, 0.04, 0.05
floyd2-p.prisem.washington.edu	GOOD	18:35:02 up 20 days, 59 min, 1 user, load average: 0.08, 0.04, 0.05
jenkins-int.prisem.washington.edu	GOOD	18:35:03 up 146 days, 8:31, 1 user, load average: 0.01, 0.02, 0.05

```
| lapp.prisem.washington.edu          | GOOD    | 18:35:02 up 146 days,  8:31, 1_
↪user,  load average: 0.16, 0.05, 0.06
↪
| eclipse.prisem.washington.edu        | DARK    | SSH encountered an unknown error_
↪during the connection. We recommend you re-run the command using -vvvv, which will_
↪enable SSH debugging output to help diagnose the issue |
| lancaster.prisem.washington.edu      | DARK    | SSH encountered an unknown error_
↪during the connection. We recommend you re-run the command using -vvvv, which will_
↪enable SSH debugging output to help diagnose the issue |
+-----+-----+-----+
↪-----+
↪-----+
```

Note: As can be seen here, the hosts `eclipse.prisem.washington.edu` and `lancaster.prisem.washington.edu` do not conform with the standard use of Ansible via SSH. These kind of *one-off* or manually-configured hosts limit the scalability and consistent use of Ansible as a system configuration and management tool.

Adding a Module in Another Repo

```
[dimsenv] dittrich@dimsdemo1:~/git/ansible-playbooks () $ cookiecutter https://git.
↪openstack.org/openstack-dev/cookiecutter.git
Cloning into 'cookiecutter'...
remote: Counting objects: 602, done.
remote: Compressing objects: 100% (265/265), done.
remote: Total 602 (delta 345), reused 563 (delta 310)
Receiving objects: 100% (602/602), 81.17 KiB | 0 bytes/s, done.
Resolving deltas: 100% (345/345), done.
Checking connectivity... done.
module_name [replace with the name of the python module]: dims_ansible_playbook
repo_group [openstack]: dims
repo_name [replace with the name for the git repo]: ansible-playbooks
launchpad_project [replace with the name of the project on launchpad]:
project_short_description [OpenStack Boilerplate contains all the boilerplate you_
↪need to create an OpenStack package.]: Python ansible-playbook module for dimscli
Initialized empty Git repository in /home/dittrich/git/ansible-playbooks/ansible-
↪playbooks/.git/
[master (root-commit) 7d01bbe] Initial Cookiecutter Commit.
26 files changed, 647 insertions(+)
create mode 100644 .coveragerc
create mode 100644 .gitignore
create mode 100644 .gitreview
create mode 100644 .mailmap
create mode 100644 .testr.conf
create mode 100644 CONTRIBUTING.rst
create mode 100644 HACKING.rst
create mode 100644 LICENSE
create mode 100644 MANIFEST.in
create mode 100644 README.rst
create mode 100644 babel.cfg
create mode 100644 dims_ansible_playbook/__init__.py
create mode 100644 dims_ansible_playbook/tests/__init__.py
create mode 100644 dims_ansible_playbook/tests/base.py
create mode 100644 dims_ansible_playbook/tests/test_dims_ansible_playbook.py
```

```
create mode 100755 doc/source/conf.py
create mode 100644 doc/source/contributing.rst
create mode 100644 doc/source/index.rst
create mode 100644 doc/source/installation.rst
create mode 100644 doc/source/readme.rst
create mode 100644 doc/source/usage.rst
create mode 100644 requirements.txt
create mode 100644 setup.cfg
create mode 100644 setup.py
create mode 100644 test-requirements.txt
create mode 100644 tox.ini
[dimsenv] dittrich@dimsdemo1:~/git/ansible-playbooks () $ ls -l
total 4
drwxrwxr-x 5 dittrich dittrich 4096 Jan  1 16:17 ansible-playbooks
```


Service Discovery Using Consul

Consul provides many services that are used by DIMS components, including a key/value store and DNS service. DIMS takes advantage of the DNS service by having `dnsmasq` on each host direct certain queries to the Consul cluster for resolution, which can be used for service discovery (as opposed to hard-coding IP addresses or specific host names and port numbers in source code or configuration files.) The chapter *Developing modules for the DIMS CLI app (dimscli)* discusses some of the ways Consul is accessed by `dimscli` (e.g., see Section *Adding New Columns to Output*)

A program named `ianitor` (GitHub [ClearcodeHQ/ianitor](#)) facilitates using this Consul DNS capability by wrapping services so they are registered in Consul's DNS and monitored by Consul's health checking features. This would allow a monitoring application to notify someone when a DIMS service component (such as something in the backend data store) becomes unavailable.

Note: The `ianitor` package from PyPi is installed in the DIMS Python Virtual Environment, so it should be available on all DIMS components that would need it.

This registration and service discovery process be illustrated using the `netcat` (`nc`) program to create a listening process that will demonstrate how this works.

First, we start `nc` on a specific listening port

```
[dimsenv] dittrich@dimsdemo1:~ () $ ianitor --port 9999 netcat -- nc -l 9999
```

There is no output at this point, since `nc` is now running in the foreground (under the watch of `ianitor`, also running in the foreground) patiently listening on port 9999 for something to connect to it. You can prove to yourself that it is running by looking in the process tree:

```
init(1)--ModemManager(1000)--{ModemManager}(1032)
      |                                     \-{ModemManager}(1036)
      | ...
      |-lightdm(1662)--Xorg(1673)
      |               |-lightdm(1738)--init(2060)--GoogleTalkPlugi(3880)--+
➔ {GoogleTalkPlugi}(3881)
      |               |               |               |               |
      |               |               |               |               |
```

```

| | | | -tmux (3066) +-bash (4512) ---
↪ ianitor (680) --- nc (683)
| | | | ...
| | | | ...

```

Now that the service is running, we can validate that `ianitor` has registered it in Consul. Figure [Consul Service Listing](#) shows Consul’s view of **Services** showing `service:netcat` has been registered and is alive and healthy.

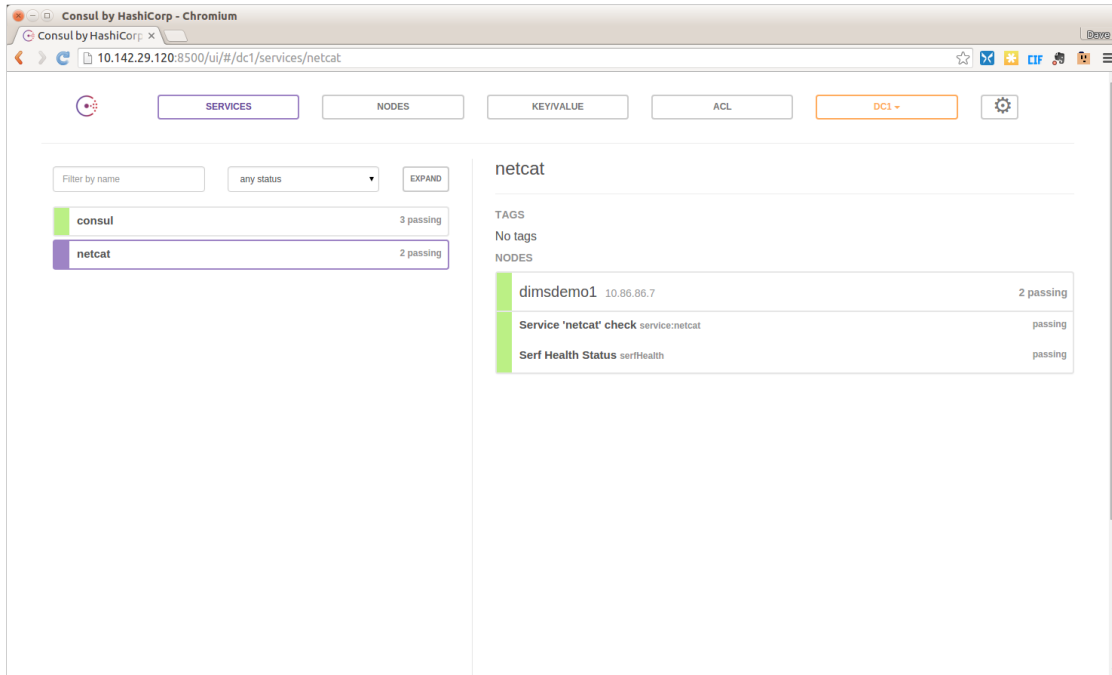


Fig. 8.1: Consul Service Listing

Using `dig`, the host on which this service was registered can be obtained by a simple **A** record lookup for `netcat.service.consul`, as seen here:

```

[dimsenv] dittrich@dimsdemo1:~ () $ dig netcat.service.consul

; <<>> DiG 9.9.5-3ubuntu0.7-Ubuntu <<>> netcat.service.consul
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16448
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;netcat.service.consul.      IN      A

;; ANSWER SECTION:
netcat.service.consul.  0      IN      A      10.86.86.7

;; Query time: 26 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Jan 24 12:19:58 PST 2016
;; MSG SIZE rcvd: 76

```

Now switch to Consul’s **Nodes** tab. Figure [Consul service registration for netcat](#) shows that node `dimsdemo1` is

running the service `netcat`, and this time the service port is also shown to the right (“: 9999”):

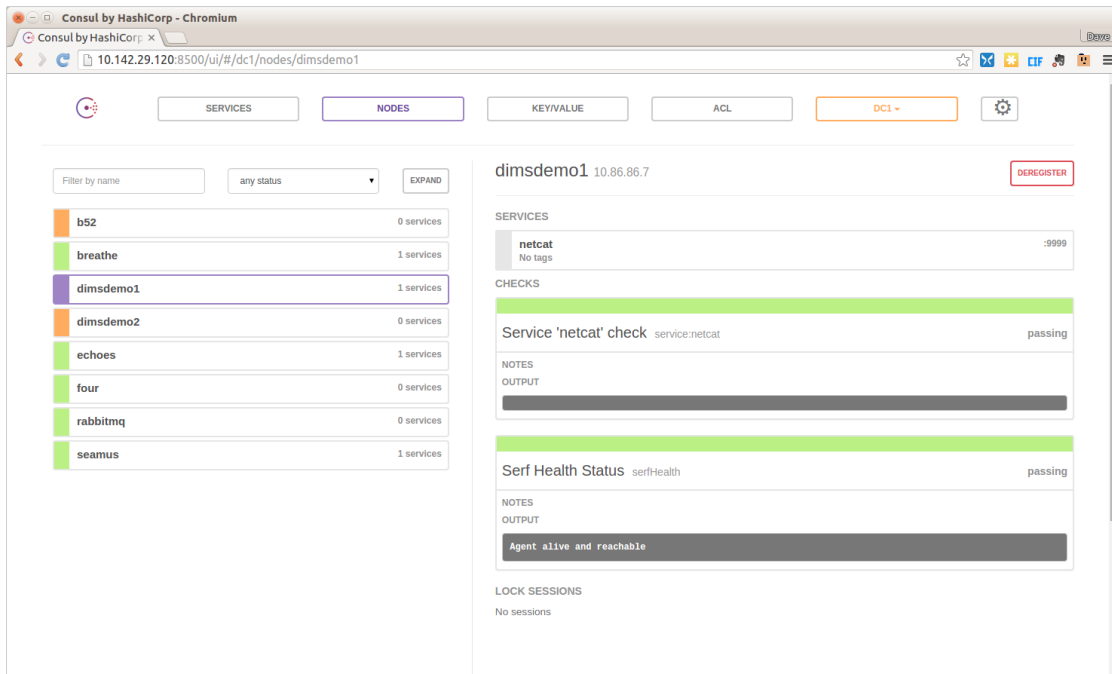


Fig. 8.2: Consul service registration for netcat

The service’s port number can also be obtained from Consul via `dnsmasq` by asking for the DNS **SRV** record for `netcat.service.consul`:

```
[dimsenv] dittrich@dimsdemo1:~ () $ dig netcat.service.consul SRV

; <<>> DiG 9.9.5-3ubuntu0.7-Ubuntu <<>> netcat.service.consul SRV
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8464
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;netcat.service.consul.      IN      SRV

;; ANSWER SECTION:
netcat.service.consul.  0      IN      SRV      1 1 9999 dimsdemo1.node.dc1.consul.

;; ADDITIONAL SECTION:
dimsdemo1.node.dc1.consul. 0      IN      A        10.86.86.7

;; Query time: 13 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Jan 24 12:48:44 PST 2016
;; MSG SIZE rcvd: 146
```

Now we can test connecting to the `netcat` listener (which will show anything that gets sent to it after the TCP connection is established.)

Attention: When attempting to duplicate this example, keep in mind that you must have already enabled `iptables` access to the port on which `nc` is listening, otherwise any connection attempt will be blocked and this won't work as shown here. **Always** keep `iptables` in mind when trying to expose network services and test them.

The first test will be using `curl` from the command line:

```
[dimsenv] dittrich@dimsdemo1:~ () $ curl --data Hello http://dimsdemo1.node.dcl.
↪consul:9999/areyouthere
```

Going back to the window where we ran `ianitor`, the result is the following:

```
[dimsenv] dittrich@dimsdemo1:~ () $ ianitor --port 9999 netcat -- netcat -l 9999
POST /areyouthere HTTP/1.1
User-Agent: curl/7.35.0
Host: dimsdemo1.node.dcl.consul:9999
Accept: */*
Content-Length: 5
Content-Type: application/x-www-form-urlencoded

Hello
```

Note: Because `netcat` simply listens on a port and then prints out what it receives (never sending anything back), both windows will hang. Just **CTRL-C** to kill them. This is just a proof-of-concept, not a real service. If you kill the `ianitor/nc` command first, the `curl` response will make this very clear with this message:

```
curl: (52) Empty reply from server
```

If you connect directly using `http://dimsdemo1.node.dcl.consul:9999` from a browser, you would get a slightly different result.

```
[dimsenv] dittrich@dimsdemo1:~ () $ ianitor --port 9999 netcat -- netcat -l 9999
GET / HTTP/1.1
Host: dimsdemo1.node.dcl.consul:9999
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/
↪43.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

In practice, `ianitor` would be used to wrap a service that is being started by some process manager, such as `supervisord`. See the [Example supervisord config](#) on the `ianitor` GitHub page.

Debugging and Development

This chapter covers some tools and tactics used for testing and debugging misbehaving system components, or obtaining sufficient detail about how subsystem components work in order to control them to achieve project goals and requirement objectives. Executing some command line or triggering an action in a user interface that results in the system appearing to “hang” can be caused by many things. Just looking at the surface and seeing no action is useless in determining the root cause of the issue. The ability to turn a “black box” into a “transparent box” is invaluable to the process of testing and debugging.

Hint: Some useful resources on the processes of *testing and debugging* are listed here:

- [Testing and Debugging](#)
 - [White-box testing, Wikipedia](#)
 - [White-Box Testing](#), by Oliver Cole, March 1, 2000
-

Determining File System Affects of Running Programs

Many programs used in the DIMS project consume gigabytes of disk storage, often in hidden locations that are not obvious to the user. The act of taking an Ubuntu 14.04 install ISO image, converting it with Packer into a BOX file, turning that into a Virtualbox image, and instantiating a Vagrant virtual machine can turn just under 1 GB into 3-5 GB of storage. Multiply that by a dozen or more virtual machines and this quickly can add up. If you are not aware of what gets created, and you change names of virtual machines, you can end up with a huge amount of wasted disk space with unused virtual disks, virtual machine images, etc.

For this reason, every programmer developing tools that use programs like this should be methodical about understanding every process in terms of inputs, process, and **outputs**, such that it is possible to **see** what is produced to help the person using your tools know what is happening, and to **undo** those effects in a controlled way to simplify cleaning up.

Hint: An easy way to help the user of your tools is to be organized and put large files related to a workflow like Packer->Virtualbox->Vagrant VM creation under a single directory path like `/vm` that can be deleted in one step, backed

up and moved to another system with a larger hard drive, or expanded by mounting a second hard drive onto that directory path as a mount point. Scattering the files across many unrelated subdirectories in random locations and depths within the user's \$HOME directory tree makes it much harder to handle a situation where the hard drive on a laptop reaches 100% utilization.

Let's take a look at a portion of the workflow of Packer->Virtualbox->Vagrant creation to see how to white-box disk utilization and space management.

We start by changing directory into the \$GIT/dims-packer repository where tools for creating Vagrants using Packer are kept. We create an initial empty file to serve as a marker in time for then locating any files that were created **after** this file.

Note: The example here will search through a set of directories that were chosen based on knowledge that they exist and are used by various tools. To obtain this knowledge, it is often helpful to start looking at the root of the filesystem (/) and look for **any** files in **any** directories, which you will quickly find has a lot of unrelated file system additions that just happen to have been made at the same time as the program you were running. A more precise way to identify where files are created is to trace execution of the program in question, following all forked children, using a program like strace and/or ltrace, however these tools require a much deeper understanding of how the Unix/Linux kernel works.

```
$ cd $GIT/dims-packer
```

```
$ cd $GIT/dims-packer
$ find /home/dittrich/.vagrant.d/ /home/dittrich/.packer.d/ /home/dittrich/
↳VirtualBox\ VMs/ . /vm -newer foo -ls
56230373    4 drwxrwxr-x    7 dittrich dittrich    4096 Mar 15 13:14 /home/dittrich/.
↳vagrant.d/
56230688    0 -rw-rw-r--    1 dittrich dittrich          0 Mar 15 13:14 /home/dittrich/.
↳vagrant.d/data/machine-index/index.lock
56230689    4 drwxr-xr-x    2 dittrich dittrich    4096 Mar 15 13:14 /home/dittrich/.
↳packer.d/
56230691    4 -rw-rw-r--    1 dittrich dittrich    318 Mar 15 13:14 /home/dittrich/.
↳packer.d/checkpoint_cache
55314574    4 drwx-----    6 dittrich dittrich    4096 Mar 15 13:24 /home/dittrich/
↳VirtualBox\ VMs/
58589344 2183628 -rw-----    1 dittrich dittrich 2240348160 Mar 15 13:27 /home/
↳dittrich/VirtualBox\ VMs/vagrant-run-ns1_default_1458069887689_42029/ns1_box-disk1.
↳vmdk
58987212    4 drwx-----    2 dittrich dittrich    4096 Mar 15 13:24 /home/dittrich/
↳VirtualBox\ VMs/devserver
55574611    4 drwxrwxr-x   19 dittrich dittrich    4096 Mar 15 13:24 .
58590167    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:24 ./vagrant-output
58590705 633044 -rw-rw-r--    1 dittrich dittrich 648229139 Mar 15 13:24 ./vagrant-
↳output/packer_devserver_box_virtualbox.box
55574679    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:14 ./ubuntu_64_
↳vagrant
55574655    4 -rw-rw-r--    1 dittrich dittrich    3263 Mar 15 13:14 ./ubuntu_64_
↳vagrant/devserver-base.json
55574654    4 -rw-rw-r--    1 dittrich dittrich    3044 Mar 15 13:14 ./ubuntu_64_
↳vagrant/devserver-box.json
58590704    4 drwxr-xr-x    2 dittrich dittrich    4096 Mar 15 13:21 ./output-
↳devserver
58590711   12 -rw-----    1 dittrich dittrich   10629 Mar 15 13:20 ./output-
↳devserver/devserver.ovf
58590712 620528 -rw-----    1 dittrich dittrich 635416064 Mar 15 13:21 ./output-
↳devserver/devserver-disk1.vmdk
```

55574612	4	drwxrwxr-x	8	dittrich	dittrich	4096	Mar 15 13:27	./.	git
----------	---	------------	---	----------	----------	------	--------------	-----	-----

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/dims-packer (feature/dims-696*) $ touch foo2
[dimsenv] dittrich@dimsdemo1:~/dims/git/dims-packer (feature/dims-696*) $ find /home/
↳ dittrich/.vagrant.d/ /home/dittrich/.packer.d/ /home/dittrich/VirtualBox\ VMs/ . /
↳ vm -newer foo2 -ls
56230373    4 drwxrwxr-x    7 dittrich dittrich    4096 Mar 15 13:33 /home/dittrich/.
↳ vagrant.d/
56230688    0 -rw-rw-r--    1 dittrich dittrich          0 Mar 15 13:33 /home/dittrich/.
↳ vagrant.d/data/machine-index/index.lock
55574612    4 drwxrwxr-x    8 dittrich dittrich    4096 Mar 15 13:33 ./.


```
git
53346305 4 drwxr-xr-x 5 dittrich dittrich 4096 Mar 15 13:33 /vm
53346306 4 drwxrwxr-x 2 dittrich dittrich 4096 Mar 15 13:33 /vm/devserver
53346314 4 -rw-rw-r-- 1 dittrich dittrich 1 Mar 15 13:33 /vm/devserver/.
↳ vagrant-IP
53346310 4 -rw-rw-r-- 1 dittrich dittrich 6 Mar 15 13:33 /vm/devserver/.
↳ vagrant-ISDESKTOP
53346311 4 -rw-rw-r-- 1 dittrich dittrich 7 Mar 15 13:33 /vm/devserver/.
↳ vagrant-VMTYPE
53346312 4 -rw-rw-r-- 1 dittrich dittrich 7 Mar 15 13:33 /vm/devserver/.
↳ vagrant-PLATFORM
53346309 4 -rw-rw-r-- 1 dittrich dittrich 10 Mar 15 13:33 /vm/devserver/.
↳ vagrant-NAME
53346313 4 -rw-rw-r-- 1 dittrich dittrich 32 Mar 15 13:33 /vm/devserver/.
↳ vagrant-BOXNAME
53346316 4 -rw-rw-r-- 1 dittrich dittrich 26 Mar 15 13:33 /vm/devserver/.
↳ vagrant-VAGRANTFILEPATH
53346319 8 -rwxrwxr-x 1 dittrich dittrich 4351 Mar 15 13:33 /vm/devserver/
↳ test.vagrant.ansible-current
53346318 8 -rw-rw-r-- 1 dittrich dittrich 4245 Mar 15 13:33 /vm/devserver/
↳ Makefile
53346315 4 -rw-rw-r-- 1 dittrich dittrich 1 Mar 15 13:33 /vm/devserver/.
↳ vagrant-FORWARDPORT
53346308 4 -rw-rw-r-- 1 dittrich dittrich 2738 Mar 15 13:33 /vm/devserver/
↳ Vagrantfile
53346307 4 -rw-rw-r-- 1 dittrich dittrich 2028 Mar 15 13:33 /vm/devserver/.
↳ vagrant_show
53346317 4 -rw-rw-r-- 1 dittrich dittrich 199 Mar 15 13:33 /vm/devserver/
↳ hosts
```


```

```
[dimsenv] dittrich@dimsdemo1:~/dims/git/dims-packer (feature/dims-696*) $ touch foo3
[dimsenv] dittrich@dimsdemo1:~/dims/git/dims-packer (feature/dims-696*) $ find /home/
↳ dittrich/.vagrant.d/ /home/dittrich/.packer.d/ /home/dittrich/VirtualBox\ VMs/ . /
↳ vm -newer foo3 -ls
56230373    4 drwxrwxr-x    7 dittrich dittrich    4096 Mar 15 13:48 /home/dittrich/.
↳ vagrant.d/
56230681    4 drwxrwxr-x    4 dittrich dittrich    4096 Mar 15 13:34 /home/dittrich/.
↳ vagrant.d/data
56232110    0 -rw-rw-r--    1 dittrich dittrich          0 Mar 15 13:34 /home/dittrich/.
↳ vagrant.d/data/lock.dotlock.lock
56230688    0 -rw-rw-r--    1 dittrich dittrich          0 Mar 15 13:48 /home/dittrich/.
↳ vagrant.d/data/machine-index/index.lock
56232608    0 -rw-rw-r--    1 dittrich dittrich          0 Mar 15 13:34 /home/dittrich/.
↳ vagrant.d/data/lock.machine-action-fab0alf680af28d59f47b677629a540a.lock
56230682    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:35 /home/dittrich/.
↳ vagrant.d/tmp
56230680    4 drwxrwxr-x   11 dittrich dittrich    4096 Mar 15 13:35 /home/dittrich/.
↳ vagrant.d/boxes
```

```

58987205    4 drwxrwxr-x    3 dittrich dittrich    4096 Mar 15 13:35 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox
58987206    4 drwxrwxr-x    3 dittrich dittrich    4096 Mar 15 13:35 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox/0
58987207    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:35 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox/0/virtualbox
58987202 646144 -rw-rw-r--    1 dittrich dittrich 661647360 Mar 15 13:35 /home/
↳dittrich/.vagrant.d/boxes/packer_devserver_box_virtualbox/0/virtualbox/devserver_
↳box-disk1.vmdk
58987203    4 -rw-rw-r--    1 dittrich dittrich        26 Mar 15 13:35 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox/0/virtualbox/metadata.json
58987200    4 -rw-rw-r--    1 dittrich dittrich       258 Mar 15 13:34 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox/0/virtualbox/Vagrantfile
58987201   12 -rw-rw-r--    1 dittrich dittrich   10785 Mar 15 13:34 /home/dittrich/.
↳vagrant.d/boxes/packer_devserver_box_virtualbox/0/virtualbox/box.ovf
55574611    4 drwxrwxr-x   19 dittrich dittrich    4096 Mar 15 13:48 .
55574612    4 drwxrwxr-x    8 dittrich dittrich    4096 Mar 15 13:48 ../git
55575296    4 -rw-rw-r--    1 dittrich dittrich    2590 Mar 15 13:48 ../make-devserver-
↳201603151348.txt
53346306    4 drwxrwxr-x    5 dittrich dittrich    4096 Mar 15 13:48 /vm/devserver
53346314    4 -rw-rw-r--    1 dittrich dittrich        14 Mar 15 13:48 /vm/devserver/.
↳vagrant-IP
53346310    4 -rw-rw-r--    1 dittrich dittrich         6 Mar 15 13:48 /vm/devserver/.
↳vagrant-ISDESKTOP
53346311    4 -rw-rw-r--    1 dittrich dittrich         7 Mar 15 13:48 /vm/devserver/.
↳vagrant-VMTYPE
53346312    4 -rw-rw-r--    1 dittrich dittrich         7 Mar 15 13:48 /vm/devserver/.
↳vagrant-PLATFORM
53346309    4 -rw-rw-r--    1 dittrich dittrich        10 Mar 15 13:48 /vm/devserver/.
↳vagrant-NAME
53346313    4 -rw-rw-r--    1 dittrich dittrich        32 Mar 15 13:48 /vm/devserver/.
↳vagrant-BOXNAME
53347678    4 drwxrwxr-x   10 dittrich dittrich    4096 Mar 15 13:48 /vm/devserver/
↳dims-keys
53347720    0 -rw-rw-r--    1 dittrich dittrich         0 Mar 15 13:48 /vm/devserver/
↳dims-keys/README.rd
53347719    4 -rw-rw-r--    1 dittrich dittrich        43 Mar 15 13:48 /vm/devserver/
↳dims-keys/.gitignore
53347722    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:48 /vm/devserver/
↳dims-keys/ansible-pub
. . .
53347752    4 -rw-rw-r--    1 dittrich dittrich        402 Mar 15 13:48 /vm/devserver/
↳dims-keys/ssh-pub/dims_andclay_rsa.pub
53347775    4 -rw-rw-r--    1 dittrich dittrich        79 Mar 15 13:48 /vm/devserver/
↳dims-keys/ssh-pub/dims_eliot_rsa.sig
53346320    4 drwxrwxr-x    3 dittrich dittrich    4096 Mar 15 13:34 /vm/devserver/.
↳vagrant
53346321    4 drwxrwxr-x    3 dittrich dittrich    4096 Mar 15 13:34 /vm/devserver/.
↳vagrant/machines
53346322    4 drwxrwxr-x    3 dittrich dittrich    4096 Mar 15 13:34 /vm/devserver/.
↳vagrant/machines/default
53346323    4 drwxrwxr-x    2 dittrich dittrich    4096 Mar 15 13:34 /vm/devserver/.
↳vagrant/machines/default/virtualbox
53346316    4 -rw-rw-r--    1 dittrich dittrich        26 Mar 15 13:48 /vm/devserver/.
↳vagrant-VAGRANTFILEPATH
53346318    8 -rw-rw-r--    1 dittrich dittrich   4245 Mar 15 13:48 /vm/devserver/
↳Makefile
53346315    4 -rw-rw-r--    1 dittrich dittrich         1 Mar 15 13:48 /vm/devserver/.
↳vagrant-FORWARDPORT

```



```

53346308    4 -rw-rw-r--    1 dittrich dittrich    2751 Mar 15 13:48 /vm/devserver/
↳Vagrantfile
53346307    4 -rw-rw-r--    1 dittrich dittrich    2041 Mar 15 13:48 /vm/devserver/.
↳vagrant_show
53346317    4 -rw-rw-r--    1 dittrich dittrich     212 Mar 15 13:48 /vm/devserver/
↳hosts

```

```

$ touch foo4
$ find /home/dittrich/.vagrant.d/ /home/dittrich/.packer.d/ /home/dittrich/
↳VirtualBox\ VMs/ . /vm -newer foo4 -ls
58589344 2183628 -rw-----    1 dittrich dittrich 2240348160 Mar 15 14:17 /home/
↳dittrich/VirtualBox\ VMs/vagrant-run-ns1_default_1458069887689_42029/ns1_box-disk1.
↳vmdk
55574611    4 drwxrwxr-x   19 dittrich dittrich    4096 Mar 15 14:13 .
55576829   28 -rw-rw-r--    1 dittrich dittrich   27191 Mar 15 14:13 ./Makefile
55574612    4 drwxrwxr-x    8 dittrich dittrich    4096 Mar 15 14:13 ./git
53870594    4 drwxrwxr-x    5 dittrich dittrich    4096 Mar 15 14:15 /vm/vagrant-run-
↳ns1
53870676    4 -rw-rw-r--    2 dittrich dittrich         4 Mar 15 14:13 /vm/vagrant-run-
↳ns1/ns1.dims
53870676    4 -rw-rw-r--    2 dittrich dittrich         4 Mar 15 14:13 /vm/vagrant-run-
↳ns1/ns1.local
53346306    4 drwxrwxr-x    5 dittrich dittrich    4096 Mar 15 13:51 /vm/devserver
53347790    4 -rw-rw-r--    1 dittrich dittrich    2756 Mar 15 13:51 /vm/devserver/
↳Vagrantfile

```


CHAPTER 10

Docker Datacenter

This chapter documents a walk thru for running a development instance of [Docker Universal Control Plane](#), part of [Docker Datacenter](#).

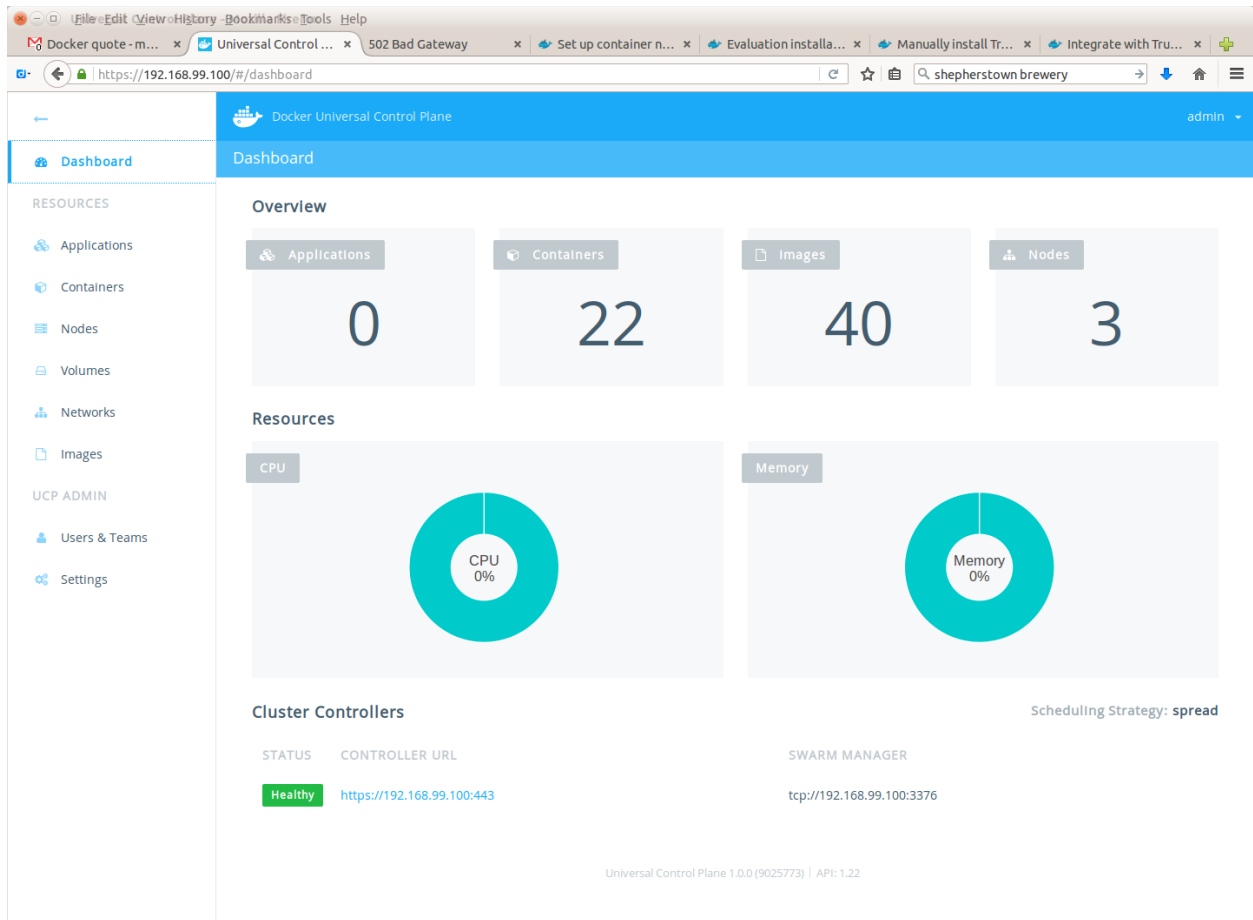
Watch a [UCP demo](#).

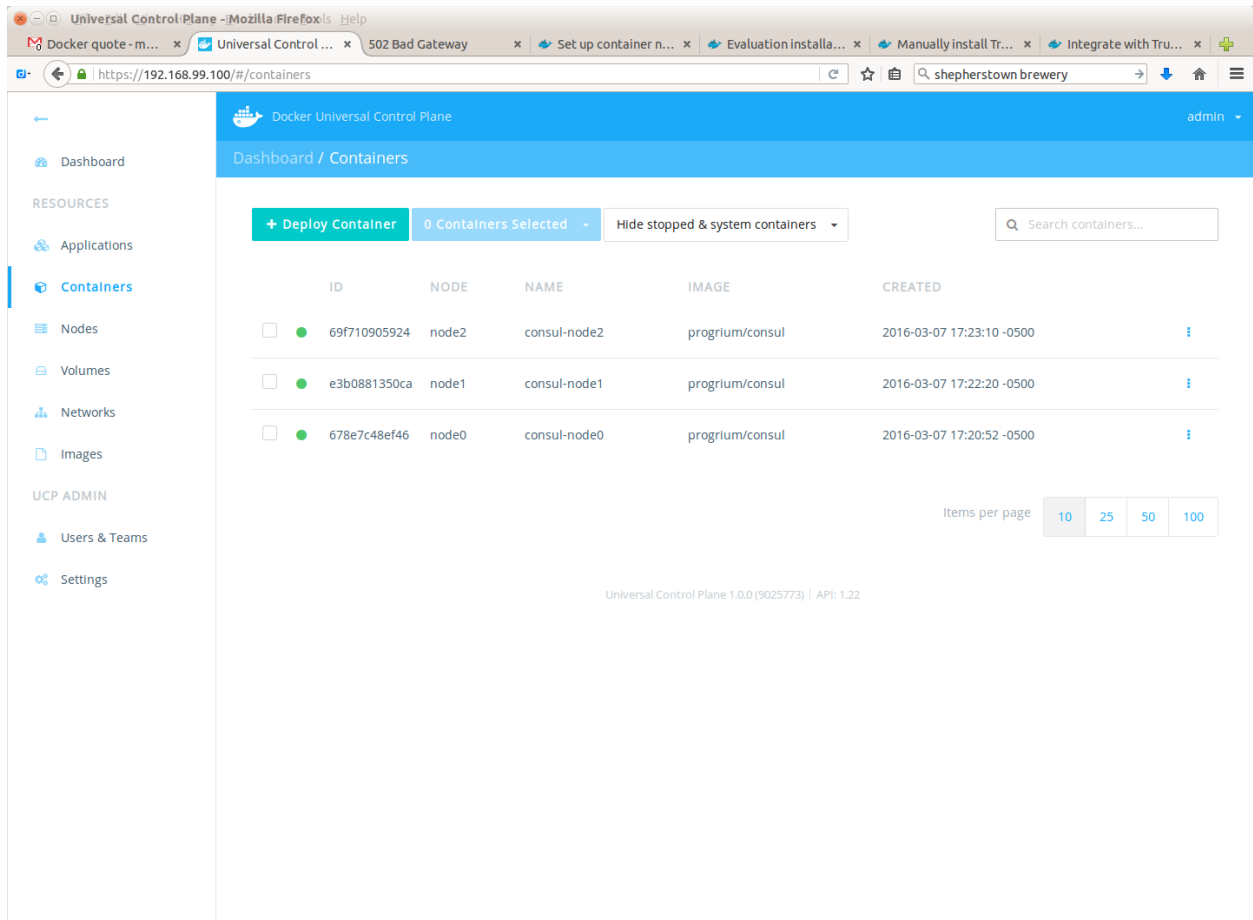
Datacenter Walk-thru

The following output walks thru these items:

- starting 3 VMs with [Docker Machine](#)
- installing UCP on one node as a controller; joining 2 other nodes
- setting up [container networking](#) on each node
- creating one [overlay network](#)
- starting a [Consul](#) container on each node

Once you’ve completed the steps outlined, you should be able to go to <https://<controller-ip>:443>, log in with “admin” and the password you gave during the prompt, submit the license, and see the following:





The screenshot shows the Docker Universal Control Plane web interface. The browser address bar indicates the URL is `https://192.168.99.100/#/containers`. The interface has a blue header with the Docker logo and the text "Docker Universal Control Plane" and "admin". Below the header is a breadcrumb "Dashboard / Containers". The main content area features a table of containers. Above the table are buttons for "+ Deploy Container", "0 Containers Selected", and a dropdown for "Hide stopped & system containers". A search bar is also present. The table has columns for ID, NODE, NAME, IMAGE, and CREATED. It lists three containers, all with the image "progrum/consul". At the bottom right, there is a pagination control showing "Items per page" with options 10, 25, 50, and 100. The footer of the interface displays "Universal Control Plane 1.0.0 (9025773) | API: 1.22".

	ID	NODE	NAME	IMAGE	CREATED	
<input type="checkbox"/>	69f710905924	node2	consul-node2	progrum/consul	2016-03-07 17:23:10 -0500	⋮
<input type="checkbox"/>	e3b0881350ca	node1	consul-node1	progrum/consul	2016-03-07 17:22:20 -0500	⋮
<input type="checkbox"/>	678e7c48ef46	node0	consul-node0	progrum/consul	2016-03-07 17:20:52 -0500	⋮

Further Information

As more is learned about Docker Datacenter, particularly admin-related information, it will be documented here.

CHAPTER 11

License

Berkeley Three Clause License
=====

Copyright (c) 2014, 2016 University of Washington. All rights reserved.

Redistribution **and** use **in** source **and** binary forms, **with or** without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions **and** the following disclaimer.
2. Redistributions **in** binary form must reproduce the above copyright notice, this list of conditions **and** the following disclaimer **in** the documentation **and/or** other materials provided **with** the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse **or** promote products derived **from this** software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Section author: Dave Dittrich dittrich@u.washington.edu

Copyright © 2014-2016 University of Washington. All rights reserved.